SOFTWARE CONFIGURATION MANAGEMENT HANDBOOK

THIRD EDITION

ALEXIS LEON

Software Configuration Management Handbook

Third Edition

For a listing of recent titles in the
Artech House Computing Library,
turn to the back of this book.

Software Configuration Management Handbook

Third Edition

Alexis Leon

ARTECH
HOUSE
BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress

British Library Cataloguing in Publication Data
A catalog record for this book is available from the British Library.

ISBN-13: 978-1-60807-843-1

Cover design by John Gomes

685 Canton Street
Norwood, MA

10 9 8 7 6 5 4 3 2 1

To my parents Leon Alexander and Santhamma Leon
for their love, encouragement, and support

Contents

CHAPTER 17

Preface

Configuration management (CM) is the art of identifying, organizing, and controlling modifications to the software being built by a programming team. The goal is to maximize productivity by minimizing mistakes. Practicing CM in a software project has many benefits, including increased development productivity, better control over the project, better project management, reduction in errors and bugs, faster problem identification and bug fixes, and improved customer goodwill. However, a single software CM (SCM) solution is not suited for all projects; while the core SCM objectives and functions remain the same, the SCM system has to be tailored to each project.

Today's software development environment is highly complex and sophisticated. At times, multiple companies join forces to develop a single product. Similarly, even within one company, it might take several geographically separate teams to develop the various subsystems of just one product or system. Managing these projects without any scientific tools could result in costly product recalls or project failures. SCM is the ideal solution for managing the chaos and confusion of software development, as its primary objective is to bring control to the development process.

This book details the SCM discipline, starting with the basics—the definition of SCM and its objectives and functions—and explaining SCM as it should be practiced in the software development process. Further, the book outlines the different phases in the software development life cycle and the role SCM plays in each phase. The book also details the pitfalls of the software development process, including the need, importance, and benefits of SCM, and demystifies the common misconceptions about SCM. In addition, the book clearly explains basic SCM concepts such as baselines, versions, variants, delta storage, branching, merging, and releases and provides in-depth coverage of the four pillars of SCM: identification, control, status accounting, and audits.

After familiarizing readers with basic terminology and concepts, the book exhaustively covers advanced topics, including the following:

- SCM implementation phases;
- Build and release management;
- Interface and subcontractor control;
- Software libraries;
- SCM plans and guidelines for writing good SCM plans;
- SCM standards;
- The role of SCM in software process improvement (SPI) models (e.g., CMM, CMMI, ISO SPICE, BOOTSTRAP, Trillium, ITIL, COBIT, and SWEBOK);

- SCM organization;
- Documentation management and control (DMC);
- Product data management (PDM).

In addition, the book covers the various SCM deployment models, from traditional to software as a service (SaaS). It also describes popular transition strategies and includes a completely new chapter on the latest development in the field of software development in the cloud, source code repositories—detailing their features, advantages, and limitations and describing how to select one that is best suited for an organization.

Subsequently, the book covers SCM tools, one of the most important aspects of SCM, explaining topics like SCM automation, the advantages of SCM tools, and pointers on tool selection so that readers can find the SCM tools best suited for their organization or project. A salient feature of this edition is its comprehensive coverage of the different activities required to plan, design, implement, operate, and maintain a good SCM system. Accordingly, the book thoroughly explains SCM system design, tool selection, implementation, and post-implementation; the operation and maintenance of SCM systems; and how to perform SCM in different scenarios (e.g., very large projects, website management, distributed environments, and integrated development environments).

The book's two appendices—the first detailing SCM resources on the Internet and the second providing a thorough SCM bibliography—will be of immense value for readers who want to further explore the new frontiers of SCM. The book also features an extensive glossary and acronym list to help readers when they encounter unfamiliar terms in the early chapters of the book.

One important aspect of this book is that it does not rely on any specific tool or standard for explaining SCM concepts and techniques. In fact, one of the main objectives of this book is to give the reader enough information about SCM and its mechanics and implementation without being tool- or standard-specific. Accordingly, the book gives information on how to select the right SCM tool for an organization or project and how to implement, manage, and maintain the tool so that the organization can reap the full benefits of SCM.

Changes in the Third Edition

The book has been revised to include the latest developments in the field of SCM. It explains the concepts of SCM, demystifies its myths and misconceptions, and gives an overview of the technologies that work with SCM systems to make organizations work at high efficiencies. The book comprehensively covers the implementation of an SCM system that is best suited for an organization—starting from package selection and continuing through tasks associated with phases such as team selection, implementation plan preparation, implementation, project management and monitoring, training, and post-implementation.

In addition, this edition includes new chapters on implementation challenges, deployment methodologies, transition strategies, and source code repositories and

completely revised and rewritten chapters on SCM tools, SCM implementation, SCM standards, and SCM certifications.

How to Use This Book

The chapters in this book are organized in such a way that the concepts of SCM are developed from the ground up. Ideally, the book should be read from start to finish. However, since such a reading plan will not suit many busy and advanced readers, I have tried to write individual chapters so that they can be read independently. If readers come across a term that is not described in the chapter, they may look it up in the glossary and continue reading. Readers who are not familiar with SCM, or who are novices in this profession, should read the book from the beginning to benefit most greatly from it.

Who Should Read This Book?

This book is written for company managers who must support SCM efforts and software project managers who must plan and design SCM systems for their projects. It is also intended for those professionals who will implement the system and those who will manage and maintain the SCM system, as well as for software developers, testers, quality assurance (QA) personnel, and all who will be affected by the SCM system. The style and approach of the book is intended to be practical rather than theoretical. It is written in an easy-to-understand and jargon-free style, so that it will become an invaluable tool in understanding the discipline of SCM and a useful guide in planning, designing, implementing, managing, and maintaining a good SCM system.

CHAPTER 1

Overview of SCM

Introduction

Computers and communications are becoming integral parts of our lives. A few decades back, communications used to be between people—one person to another. Now, however, inanimate objects are getting into the act: Books can tell cash registers how much they cost; identity cards can tell door locks whether to open or not; automated guided vehicles can tell their host computer where they are in the shop floor, what they are carrying, and when they will be free; missiles can compare the landscape with their own map and hit targets with pinpoint precision; and on the Internet, people engage in lively discussions and play games even if they are physically in different continents.

The prime mover behind this digital revolution is computer software. Today software touches and controls almost all aspects of our life: Software makes us more efficient and productive; software helps people to learn and teach better; software makes entities including our homes, banks, and organizations more secure; software helps doctors to better diagnose diseases and find better treatments; software controls mission-critical applications and equipment. The list is endless.

As software becomes more and more prominent, the task of developing the software is becoming more and more difficult. Because software is used for critical applications and controlling sophisticated equipment and systems, even a small mistake or error can have catastrophic consequences. Today's software projects are becoming more and more complex—in size, sophistication, and technologies used. Now most software products cater to millions of users, support different national languages, and come in different sizes and shapes (e.g., desktop, standard, professional, and enterprise). For example, operating systems, word processors, and even enterprise resource planning (ERP) packages support multiple languages and multiple currencies. Almost all application software products (such as word processors, ERP packages, and even SCM tools) support more than one hardware or software platform. For example, we have ERP systems that run on mainframes and client-server systems; different versions of Web browsers for the PC and Mac; and database management systems that run on a variety of operating systems including MVS, UNIX, Windows NT, and Linux. Competition and advancements in technology are driving software vendors to include additional functionality and new features in their products just to stay in business.

The emergence and growing popularity of cloud computing has created new software development models and practices. With cloud computing, programmers can work on a software project from anywhere in the world if they have a computer

and an Internet connection and they can work securely because of developments in the field of computer security. Application development in the cloud has created the need for newer models for software development and support functions like SCM and software quality assurance (SQA). Since applications are being developed in a virtual environment, the computing environment and software development models have to scale up to face the new challenges of speed, 24/7 availability, and security, among others.

In addition, users of software systems have matured, and the bugs and defects in a system are detected and publicized faster than ever—thanks to the Internet. In today's software development environment, where communication facilities are advanced, the news that a software product is bad and has bugs can spread very fast. This is evident from the newsgroup postings and the news alerts that one gets so often nowadays. as a result, if the company has to save face and prevent its market share from dropping, it has to provide fixes and patches very quickly. The time that a company gets to do damage control—that is, find the cause of the bug, identify the problem area and fix the bug, ensure that the bug fixing has not created additional bugs, do regression testing, and get the bug-fixed version of the software to the customer—is much shorter than before. Companies must react very quickly in order to keep their reputations intact.

Thus, today's software development environment is very complex and the reaction or response times are very short. Millions of software professionals around the world are developing complex software systems. These systems consist of a myriad of components, each of which evolves as it is developed and maintained. The task of managing a software project successfully and delivering a high-quality and defect-free product on time and without any cost overrun is nearly an impossible task. To survive in this brutally competitive world, organizations need some sort of mechanism to keep things under control, or total chaos and confusion will result and threaten to create product or project failures and even to put organizations out of business. One such mechanism is a properly implemented SCM system. According to Whitgift [1], SCM ensures that the development and evolution of the different components of a system are efficient and controlled, so that the individual components fit together as a coherent whole.

SCM is a method of bringing control to the software development process. As Babich [2] has stated, "… On any team project, a certain degree of confusion is inevitable. The goal is to minimize the confusion so that more work can get done. The art of coordinating software development to minimize this particular type of confusion is called CM. CM is the art of identifying, organizing, and controlling modifications to the software being built by a programming team. The goal is to maximize productivity by minimizing mistakes."

SCM is a process used for more efficiently developing and maintaining software, which is accomplished by improving accountability, reproducibility, traceability, and coordination. All the functions of SCM—including the identification of configuration items, the documentation of characteristics, and controlling change—is for the purpose of assuring integrity, accountability, visibility, reproducibility, project coordination and traceability, and formal control of system and project evolution [3]. This chapter introduces the discipline of SCM.

Common SCM Myths

There are a lot of myths that surround the discipline and practice of SCM. Very often, people are not willing to adopt SCM because of these wrong notions. This section discusses some of the most common myths about SCM and attempts to demystify them.

SCM Means More Work and Procedures

Properly implementing and managing an SCM system is not an easy task. The transition period from an environment with no SCM or a rudimentary manual SCM to an SCM tool is difficult, since new skills have to be learned, and new procedures and processes have to be followed—among other challenges. Many employees think that SCM will add more work, make work more difficult, and force unnecessary procedures. Implementing and transforming to an automated SCM system is a difficult process. However, if the management and the implementation teams do their job properly, ensuring that employees are told what to expect and given proper training, then the transition can be smooth. Once employees get used to the new system, they will understand the potential benefits and the effort saved through automation of tasks and jobs and elimination of errors. Today's SCM tools automate almost all of the repetitive, monotonous, and tedious procedures, tasks, and processes, thus making the life of employees a lot easier.

SCM Will Change Current Practices and It Will Create Product Failures

The SCM package comes with proven and time-tested best practices that have been collected from the industry and integrated into the system by SCM vendors. Some of the business practices of the organizations where the SCM packages are implemented will be different from the best practices in the SCM system. One way to solve this problem is with customization—changing the code to make the SCM system function as an existing company practices. This will help an organization to continue doing business as it was previously. The downside of customization is that the customized code is not eligible for free upgrades from the vendor. This can complicate and substantially increase the cost of implementation and maintenance, making upgrades a great deal more difficult and expensive. These changes also run the risk of making your company and solution a "community of one without any other companies running a similar SCM system, making maintenance of the system nearly impossible." The better option is change the organization's business practices and adopt the best practices from the SCM package. This way, the company will improve its efficiency, streamline its business process, and avoid the headache of customization.

SCM Is a Difficult, Monotonous, and Time-Consuming Activity

Properly implementing and managing an SCM system is not an easy task. During the early days of SCM, where all the CM tasks had to be done manually, CM was a difficult, monotonous, and time-consuming activity. Even then, however, the

benefits of having an SCM system far outweighed the difficulties. Today, with the availability of sophisticated SCM tools, managing an SCM system is a totally different ball game. Today's SCM tools automate many of the repetitive, monotonous, and tedious SCM activities, significantly simplifying the practice of SCM.

SCM Is the Responsibility of Management

SCM is the responsibility of all the people involved in the software development process. Management is not responsible for the day-to-day operation of the SCM system. Management's main job is to create an organizational environment in which SCM can thrive—in other words, to give SCM its full backing. Management should also be involved in the development of SCM policies and usage guidelines, the allocation of budget, the selection of tools, and the appointment of competent professionals to implement and manage the SCM system. An SCM team needs the full backing and support of management to be able to implement the system smoothly. Accordingly, management should monitor the implementation and operation of the system, review the progress and status periodically, and take the necessary corrective actions if required. Management should also ensure that the SCM team gets the support and cooperation of all the departments.

SCM Is Just for Developers

Software development teams constitute one of the major users of SCM systems. Moreover, developers benefit the most from a properly implemented SCM system. Problems such as missing source code, the inability to find the latest version of a file, corrected mistakes reappearing, programs that suddenly stop working, and missing requirements can be avoided if a good SCM system is in place. However, many developers see SCM as a waste of time and do it only because they are forced to. This hostility toward SCM can be eliminated if developers are properly educated in SCM principles and made aware of the benefits of SCM systems. With today's SCM tools, the level of automation that is achieved is phenomenal, and performing SCM activities is not difficult. Although the development team is one of the main beneficiaries of a good SCM system, many others stand to profit, including testers, QA personnel, maintenance teams, and support teams.

SCM Is Just for the SCM Team

The days of full-fledged SCM teams are gone. Today, SCM tools have replaced humans in most areas, and most of the SCM activities have been automated. Even in cases and places where human intervention is required, the tools make the job easier by automating tasks, providing relevant information for better and faster decision-making, and enabling better communication between the people involved. Currently, the major tasks SCM teams have to perform include SCM plan preparation, tool selection, implementation and operation, user training (training on both SCM concepts and SCM tool operation), change management, build and release management, audits and reviews, and management reporting. To manage an efficient

and effective SCM system that will help the organization to develop world-class software without any problems, SCM teams need the full support and cooperation of all the other departments, including developers, testers, QA personnel, technical support staff, management, and customers.

SCM Is Just for the Maintenance and Technical Support Team

Once an SCM system is developed and tested, it is released to users. From this point onward, the maintenance phase starts. Once people start using the system, they will find many errors unnoticed during testing. Users might also ask for new features and enhancements. It is the responsibility of the maintenance team to attend to these requests and to fix the bugs that are found. The job of the maintenance and technical support teams will be a lot easier if the project followed good SCM practices. The maintenance team can fix bugs quickly if the SCM system has documented the change history, build history, details of bug fixes during development, and other such information. Also, managing and maintaining the different versions of the same product and providing support to these different versions are not possible without an SCM system. A good SCM system produces a repository of the errors that occurred during development and describes how they were fixed. As the maintenance phase continues, more and more problems and their solutions will be added to this repository, helping the technical support team to avoid the "reinventing the wheel" phenomenon. Conversely, for projects that have not followed any SCM procedures, the job of maintaining the system can turn into one of the most difficult assignments that software professionals face. A good example of this is the Y2K projects, where people had to fix problems in programs that were more than 20–30 years old, programs that had no documentation and that were developed without any programming standards and naming conventions. In conclusion, for the effective and efficient functioning of maintenance and technical support teams, a good SCM system should be in place from the very beginning of the software development process and not just at the beginning of the maintenance phase.

SCM Will Make Many Employees Redundant and Jobless

Another popular misconception about SCM systems is that their implementation will make many jobs redundant (because of automation) and that hence, many employees will lose their jobs. Although properly implemented SCM systems will automate many tasks, they will not necessarily make people redundant. Yes, there will be changes in job descriptions and in the activities people used to do. Many tasks will indeed be automated, and this will make the people who were doing those jobs unnecessary. At the same time, however, SCM systems create new job opportunities, and the very same people whose jobs have been automated can fill these new positions after receiving proper training. Here, the amount of planning that goes into relocation and retraining of employees by the management and the implementation teams can go a long way in reducing the anxiety of employees. SCM is a people system made possible by computer software and hardware.

SCM Slows Down the Software Development Process

Before the advent and popularity of SCM tools, SCM was a time-consuming process as almost all its activities were performed manually. Even in those days, however, SCM was considered worth the effort as it saved a lot of time otherwise spent in activities such as searching for missing files, fixing bugs that were already fixed, and troubleshooting problems due to the use of incorrect versions. Thus, even though SCM was a time-consuming process that slowed down developers a little, the benefits far outweighed the costs. Today, on the other hand, with the popularity of excellent SCM tools that automate almost all the SCM functions, the practice of SCM is a lot easier, and the argument that SCM slows down the software development process or decreases development productivity is totally absurd. In fact, in managing today's complex software projects SCM is a must, and any organization that attempts to develop software products without a good SCM system will be shooting themselves in the foot.

SCM Is Just To Get Certifications Like ISO and CMM

It is true that to get certifications like ISO (International Organization for Standardization), CMM (Capability Maturity Model), and the like, a good SCM system is a must. However, if SCM systems are implemented and managed just for the sake of getting such a certification, then it is more likely that the systems will not deliver the expected results. Organizations should create an environment or work culture that treats SCM as a fundamental necessity and not as necessary evil to achieve some certification or to get a contract. The SCM system will not deliver its full potential if the people involved are doing the SCM activities for the sake of doing them. In this aspect, the training of users about the benefits of SCM, SCM concepts, and efficient use of SCM tools is very important.

SCM Tools Will Take Care of Everything

It is a fact that the SCM tools have evolved over time and have become very sophisticated. Today's SCM tools automate most of SCM activities, making the life of users a lot easier. In many cases, SCM tools are integrated into the development environment so that SCM becomes a part of the development process. However, thinking that SCM tools will take care of everything can be a recipe for disaster. In fact, many SCM activities *need* human intervention and judgment, including change management, build and release management, and configuration audits. While SCM tools make these jobs easier, there is no substitute for human intelligence and decision-making. For example, an SCM tool can automate almost 80% of the change management process. However, SCM personnel should decide when to initiate a change request, when to do the impact analysis, and when to incorporate the change.

One SCM Tool Will Suit Everybody

There are hundreds of SCM tools available in the marketplace. These tools differ in features, capabilities, size, functionality, price, technical support, customizability, and scalability, among other factors. Organizations also differ from one another,

each having their own characteristics and identity. Accordingly, assuming that one tool will be suited for all organizations is wrong. Selecting and purchasing an SCM tool without analyzing whether the tool is suited for the organization might lead to disastrous consequences. For an SCM implementation to be successful, the tool that is implemented should be compatible with organizational culture, practices, and procedures. Thus, organizations must give proper attention to the selection of SCM tools that best suit them.

SCM Is Very Expensive

SCM tools come in all shapes and sizes. Some are free; some cost a small fortune. Similarly, some perform rudimentary change management activities, while others cover the entire life cycle of the development process. Many free SCM tools are available, but they have limited functionality and features and offer no technical support.

The sophisticated and high-end SCM tools are very expensive. Furthermore, an SCM system needs people to manage it. Thus, implementing and managing a SCM system is an expensive affair. However, these expenses should be weighed against the benefits of SCM systems. Efficient SCM systems increase the productivity of human resources (e.g., developers, testers, QA personnel, auditors, and managers), shorten development and change cycles, streamline software builds, reduce errors by automating monotonous and repetitive tasks, speed up recovery to previous versions of software when errors are identified, enable better management of projects by providing quality information, and improve customer satisfaction by resolving bugs and problems quickly—among other benefits. When the advantages of the SCM system are considered, it becomes evident that money spent on SCM is well-spent and that, moreover, SCM systems pay for themselves.

Once the SCM Implementation Is Complete, There Will Be No Additional Expenses

The SCM implementation is never really over. Maintenance activities—like upgrading software as new versions come out, upgrading hardware as new technology becomes available, renewing support contracts with vendors, training new employees, and conducting refresher courses for current employees—require money. In fact, the key to successful SCM is a strong SCM strategy, not just for implementation, but to keep it operating, growing, and adapting with the company—so spending on SCM never really ends. After the initial deployment, an organization has covered the major portion of the costs, such as the initial cost of purchasing the software and hardware, the cost of setting up the SCM system, and the fees for the consultants and vendor support team. However, maintenance and operation of the SCM system will continue as long as the SCM is running. Accordingly, organizations need to budget for this to prevent problems stemming from working with outdated software and hardware and untrained employees.

SCM Is Just for the Source Code

Many software professionals believe that CM is a solution suitable only for managing source code. This is far from the truth. Many of today's CM systems are

capable of managing a wide variety of digital and electronic artifacts. As we look back on how this narrow-minded view of CM developed and why it still persists, there is no great mystery. First-generation CM systems were capable of versioning only ordinary text files and had self-limiting names like source code control system (SCCS). In fact, Microsoft's CM system still has such a name—Visual SourceSafe [4]. However, CM has many other uses than just managing the source code. SCM can and should be used to manage all the project artifacts including requirements, visual models, prototypes, executables or binaries, system files, libraries, documents, development tools, test scripts, and images.

SCM Is Change Management and Defect Tracking

While change management and defect tracking are two major activities of SCM, there are many other SCM functions, including configuration identification, status accounting, and configuration audits. Nevertheless, it is true that there are tools available in the marketplace that do nothing but change management and defect tracking. These tools help project managers to bring some amount of control to the development process. In many small projects, in fact, full-fledged SCM systems are not necessary, as the cost of the system will not justify its purchase. In such cases, a change-management or defect-tracking tool can be used quite efficiently to perform and automate the change-management functions, while other SCM activities are performed manually. However, in the case of large, complex, and mission-critical projects, a simple change-management tool will not be enough. In these cases, a full-fledged SCM tool, one that covers the entire functionality of SCM, should be used.

Software Development Can Succeed Without SCM

In the early days, software development was done without SCM. Even today, software development can sometimes succeed without SCM in the case of very small projects with small project teams. However, today's software products are becoming more and more complex in size, sophistication, and technologies used. Also, the different components of a software product are not necessarily built by a single group. This is the era of multisite, distributed development where different components of a system are developed by different groups situated in different parts of the world. In such scenarios, managing a software project is a very complex task. If proper control mechanisms and procedures are not in place, software development can quickly get out of control and projects can fail. The history of the software industry is full of such software disasters, and if one closely examines these major failures, many can be attributed to the absence of an efficient SCM system.

SCM Is Just To Impress Customers

It is true that a properly implemented SCM system can help serve customers better, as it helps organizations to react faster, respond better, and deliver high-quality products at astonishing speeds. This improved efficiency and quality will go a long way in improving customer goodwill and customer relations. So, with an SCM

system you get more satisfied customers, but that is only one of the advantages of an SCM system.

A Brief History of SCM

CM has its origins in the manufacturing industry, more specifically in the U.S. defense industry. When the products that were developed were small and the product sophistication level lower, the activities of product development and design change during the entire product life cycle could be managed by a single person or a group of close-knit people. However, when products began to become more complex—as embodied by such machinery as fighter planes, tanks, and guns—it was impossible for a single person or group to maintain control over the design and production and, more importantly, the design changes. Moreover, the development of these products spanned many years and was handled by more than one person, so when control was transferred from one person to another, the associated information was lost, because no formal methods existed for documenting the design and the changes made to it.

Consequently, in 1962, the U.S. Air Force responded to the control and communication problems in the design of its jet aircraft by authoring and publishing a standard for CM, AFSCM 375–1. This first standard on CM [5] was followed by many standards, mainly from the U.S. military and the Department of Defense (the MIL and DoD standards).

As computers became popular, the importance of and focus on the software development industry began to increase. More and more people began to use computers and software products. Software systems made life easier by automating many tasks that had previously been done manually. As people got used to the convenience of automated systems, they began to demand more and more features. Software development organizations were left with no choice as more and more players entered the market with newer and better products. Thus, computer programs became increasingly complex in size, sophistication (used for mission-critical applications), and technologies used (today including workflow automation, groupware, Internet, and e-commerce). Also, the size and the nature of development teams have changed. Now, development teams consist of thousands of people, from different cultural and social backgrounds, and the various subsystems of a system could easily be developed by different companies from around the world.

As computer programs became more and more complex and difficult to manage and as computer project teams became larger and more distributed, the problems that plagued production engineers—such as the inability of a single person to control and manage the development process, the difficulty in managing change, communication breakdowns, and the difficulty in transferring knowledge when transferring responsibility—began to appear in the software development processes also. The U.S. DoD and several international organizations, including the Institute of Electrical and Electronics Engineers (IEEE), the American National Standards Institute (ANSI), and ISO, all started to address the problem of CM in the software development process and came out with their own standards. Among these, the

most widely used standards are the ANSI/IEEE standards. (Chapter 11 discusses the different standards in more detail.)

Today, SCM is accepted as a discipline and is practiced by most, if not all, software organizations, and awareness of the need and importance of SCM is increasing. Hundreds of tools and packages are available that help automate the SCM process and make the practice of SCM easier.

SCM: Concepts and Definitions

Proper application of SCM is a key component in the development of quality software. Uncontrolled changes to the software under development are usually a significant cause of changes to a project's schedule and budget; in fact, unmanaged change is the largest single cause of failure to deliver systems on time and within budget.

Bersoff, Henderson, and Siegel [6] have defined SCM as the discipline of identifying the configuration of a system at discrete points in time for the purposes of systematically controlling changes to this configuration and maintaining the integrity and traceability of this configuration throughout the system life cycle.

The IEEE [7] defines CM as a discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

What does this definition mean? First, SCM is a discipline—a discipline applying technical and administrative direction and surveillance. The term discipline refers to a system of rules, so the practice of SCM cannot be done according to the whims and fancies of individuals—it has to follow a set of rules. These rules are to be specified in a document called the SCM plan (discussed later in this chapter). The rules should be applied in a technical and administrative framework, and monitoring (surveillance) should be constant to ensure that the rules are followed. This means that SCM needs an organizational setup for carrying out the technical and administrative monitoring. SCM requires one group of people to carry out different SCM functions and another group to monitor whether the SCM activities are performed according to the rules. The size and organizational structure of this group—the SCM organization or SCM team—will vary with the size and complexity of the project.

Second, the SCM function should identify the configuration items and document their functional and physical characteristics. IEEE [7] defines a configuration item as an aggregation of hardware, software, or both that is designated for CM and treated as a single entity in the CM process. So, the SCM discipline must identify the components (e.g., documentation, programs, functions, component libraries, and data) of a software system. Then, it should document the components' functional characteristics, such as what they are supposed to do, performance criteria, and features as well as the physical characteristics such as size, number of lines, and number of modules, functions, and libraries.

Once the configuration items are identified and their characteristics documented, the SCM system should control the changes to those characteristics. This means

that once the SCM system is in place, any change to a configuration item should take place in a controlled way. Control does not mean prevention. It means that the SCM system should institute procedures that will enable people to request a change or an enhancement to a configuration item. Well-defined methods should be in place for evaluating these requests, studying the impact of each request on other configuration items, and then carrying out the changes if appropriate. In other words, the SCM system should ensure that no changes are made to the configuration items without proper authorization.

Third, the SCM system should record the change management process and report it to all those who are involved. This necessitates the documentation of the change management process. The status of the change requests should be tracked and recorded from the point of origin until completion. Processes such as change requisition, evaluation, impact analysis, and decisions on whether to implement a change should be documented and reported to all people involved.

Last, there should be some mechanism to verify that the system that is being developed and delivered is the one that is specified in the requirements definition and other related documents. In other words, the SCM system should ensure that what is developed and delivered is exactly what was required and specified. For this there should be some sort of an auditing or verification mechanism.

So, translated into plain language, SCM requires the components of a software system to be identified and their characteristics (both functional and physical) to be documented. Once this is done, then any changes to these items should only be made through proper channels. This means that somebody without proper authorization cannot make changes to an item. The entire process of change management should be documented and reported to all those who are involved. A mechanism should exist for checking and verifying that the system is being developed in accordance with the specifications.

IEEE [7] divides the SCM functions into four activities: configuration identification, configuration control, status accounting, and audits and reviews. Configuration identification is an element of CM, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation. Configuration control is the element of CM consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items. Status accounting consists of the recording and reporting of information needed to manage a configuration efficiently. Auditing is carried out to ensure that the SCM system is functioning correctly and to ensure that the configuration has been tested to demonstrate that it meets its functional requirements and that it contains all deliverable entities. We will look at these four SCM functions in more detail in Chapters 6 through 9.

Aiello and Sachs [8] define six core functional areas for CM: source code management, build engineering, environment configuration, change control, release engineering, and deployment. These six functions together cover all the CM tasks, which include the following:

- Identification and control of every configuration item;
- Selection of the exact version of each configuration item that is part of a build;
- Management and maintenance of the development;

- Checking the test, integration, and production environments to ensure that there is a formal mechanism for making changes to the configuration items;
- Packaging and identification of all configuration items of a particular release;
- Releasing the packaged releases to production.

According to Davis [9], effective SCM is not just having a tool that records who made what change to the code or documentation and when. It is also the thoughtful creation of naming conventions, policies, and procedures to ensure that all relevant parties are involved in changes to the software. SCM is not just a set of standard practices that applies uniformly to all projects. SCM must be tailored to each project's characteristics—including the size of the project, its volatility, the development process, and the extent of customer involvement. The best place to record how SCM should be performed for each project is in the SCM plan. The SCM plan documents what SCM activities are to be done, how they are to be done, who is responsible for doing specific activities, when they are to happen, and what resources are required. Chapter 13 details SCM plans, including their organization and contents.

Importance of SCM

Poor CM or lack of it often causes the most frustrating software problems. Some examples of these problems are missing source code; changed component libraries; the inability to determine what happened to a particular program or data; the inability to track why, when, and who made a change; and difficulty in finding out why programs that were working suddenly stop working. The problems are frustrating because they are very difficult to fix, and they often occur at the worst possible times [9]. Consider, for example, scenarios in which a difficult bug that was fixed suddenly reappears, a program that was working mysteriously stops working, a developed and tested feature is missing, the updated version of the requirements document is not found, or the source code and the executable program are different versions.

SCM helps reduce these problems by coordinating the work and effort of many different people working on a common project. SCM plays an important role in the software development process—analysis, design, development, testing, and maintenance—by ensuring (through configuration audits) that what was designed (as specified in the characteristics document) is what is built.

The key role of SCM is to control change activity so that problems (e.g., communication breakdown, shared data, simultaneous updates, and multiple maintenance) can be avoided. This does not mean that all the other SCM functions such as configuration identification, status accounting, and configuration audits are not important. All those activities *are* important. The configuration identification should be performed well to manage the changes effectively. The status accounting information is used by the project leaders and managers to keep the project under control. The configuration audits are required to ensure that what is specified is what is delivered. Even though we can consider configuration control (or changer

management) as the first among equals, it should be kept in mind that the other functions are also very important for the efficient and effective functioning of SCM.

SCM is not easy; one has to do a lot of work to keep an SCM system in good shape. The effort is worth it, however. Only when problems begin to crop up do users realize the importance of SCM, but by then, it is too late, and getting a project back on track can be a very tedious task without SCM.

For an SCM system to work, the people who are involved must be convinced of the importance and benefits of SCM. If SCM is done for the sake of doing it, then SCM will fail to deliver on its promises. If SCM is done just to get some certification and not in its true spirit, then it will definitely fail. If SCM is treated as a management tool or a contractual obligation, it can easily become a bureaucratic roadblock that impedes the work [10]. Another point that should be noted here is that poor SCM practices tend to ripple through an entire project, having an adverse effect on a large number of people and their work. Hence, it is important to have a good SCM system. The presence of an SCM system that is planned poorly, implemented haphazardly, and practiced inefficiently will not improve the software development process; in fact, it will create more problems than it will solve. So the SCM system should be carefully designed, properly implemented, and practiced systematically and willingly.

Benefits of SCM

A properly designed and implemented SCM system has a number of benefits. Some of the benefits are improved software development productivity, easier handling of software complexity, improved security, higher software reuse, lower software maintenance costs, better quality assurance, reduction of defects or bugs, faster problem identification and bug fixes, process-dependent development rather than person-dependent development, and assurance that the correct system was built. Chapter 4 discusses these benefits at length.

Summary

If designed properly, implemented judiciously, and used efficiently, SCM systems will raise the productivity and profits of companies dramatically. For this to happen, personnel should be educated on the potential benefits of SCM, its capabilities, and how it can help improve developmental productivity.

Many myths surround SCM. Many people consider it to be a bureaucratic process and additional work. These concerns were true to some extent in the case of manual SCM systems, but today's SCM tools automate most of the SCM functions and make the practice of SCM easier and painless.

Poor SCM practices tend to ripple through an entire project, having an adverse effect on a large number of people and their work, so it is important to have a good SCM system.

References

[1] Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

[2] Babich, W. A., *Software Configuration Management: Coordination for Team Productivity*, Boston, MA: Addison-Wesley, 1986.

[3] Ben-Menachem, M., *Software Configuration Guidebook*, London: McGraw-Hill International, 1994.

[4] Capasso, R., "Configuration Management—It's Not Just for Source Code," *The Rational Edge* (http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/feb01/ConfigurationManagementFeb01.pdf), 2001.

[5] Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, 1992.

[6] Bersoff, E. H., V. D. Henderson, and S. G. Siegel, *Software Configuration Management: An Investment in Product Integrity*, Englewood Cliffs, NJ: Prentice-Hall, 1980.

[7] IEEE, *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610–1990), IEEE Software Engineering Standards Collection (CD-ROM Edition)*, Piscataway, NJ: IEEE, 2003.

[8] Aiello, B., and Sachs, L., *Configuration Management Best Practices*, Upper Saddle River, NJ: Addison-Wesley, 2011.

[9] Davis, A. M., *201 Principles of Software Development*, New York: McGraw-Hill, 1995.

[10] Humphrey, W. S., *Managing the Software Process*, New York: Addison-Wesley, 1989.

CHAPTER 2

The Software Development Process

Introduction

The software development process is that set of actions required for efficiently transforming a user's need into an effective software solution. Efficiency means doing things in the right way, and effectiveness is doing the right things. So, for the software development process to succeed, one not only has to do the right things but also do them in the right way.

Humphrey [1] defines the software development process as the set of tools, methods, and practices that we use to produce a software product. The software development process defines the activities required for building the software systems and incorporating the methods and practices to be adopted. It also includes the activities essential for planning the project, tracking its progress, and managing the complexities of building the software. Scientific software development—also known as software engineering—uses scientific and management techniques and productivity improvement tools for developing the software.

There is no universally accepted definition for software engineering. According to Jones [2], software engineering is a methodology that uses a set of recognized criteria (e.g., functionality, reliability, and timeliness) and that has its foundations in such disciplines as computer science, mathematics, engineering, and management. The practice of software engineering is a discipline with a defined process for software development and maintenance. Software engineering aims at developing a full product that goes well beyond a small program and that uses a set of tools and techniques for improving the productivity and quality of work.

So, software engineering is not just programming, nor is it the development of a small program. It is the process of developing a software system or product. It demands management skills, communication ability, analysis, and design skills. Moreover, it means following standards and procedures and working as a team.

Software engineering is not an art. A software engineer is constrained by user requirements, team decisions, and management instructions, and the software product is the fruit of the entire team's effort rather than the creation of an individual. Also, the scope of software engineering extends far beyond the development of the software product. It involves marketing, maintenance, and after-sales support. Just because the product you have developed is the best does not mean it is going to succeed. Discipline, teamwork, marketing, money management, planning, and many other nontechnical skills play a vital role in the success of a software product or system. Thus, the objective of software engineering is not limited to the development

of a high-quality product, but includes the tasks of successfully marketing and maintaining the product.

A software product starts its life as an idea or concept. Software can be of two types—generic products (products that are produced by a software development organization and sold in the open market) and customized products (products that are developed to meet the specific needs of a customer). Examples of generic products [also known as shrink-wrapped products, packaged software, or commercial off-the-shelf (COTS) products] include word processors, electronic spreadsheets, database management systems, imaging tools, and Web browsers. Customized products are developed for a specific person or organization to meet a specific need [e.g., computerization of the operations of a bank, development of an airline or railway reservation system, and development of a software tool to accomplish a specific task (such as test data generation and code generation)]. In the case of generic products, the organization that develops the software controls the specification. In the case of custom products, the client or organization for which the product is being developed usually controls the specification. In other words, the characteristics and features of generic products are market-driven (influenced by information drawn from tools such as market research, surveys, and demos), while the client decides those of customized products.

Irrespective of the type of the software product and the way in which the software product idea was conceptualized, the software product goes through a series of development phases. In most cases, the product's features and functions are specified, then designed and implemented. The product is then put into operation, and while it is operational, it is maintained. Finally, when the usefulness of the product is over or when the product becomes obsolete, it is decommissioned. The series of steps through which the software product goes through (from conceptualization until retirement) is called the software development life cycle (SDLC).

Software Development Life Cycle (SDLC)

Every software product has a lifetime—it starts its life as a response to a user need or as a new product concept and ends up being obsolete. The life span of software systems varies from product to product. During its lifetime, software goes through various phases.

The *IEEE Standard Glossary of Software Engineering Terminology* (IEEE Std-610-1990) defines SDLC as the period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, a requirements phase, a design phase, an implementation phase, a test phase, an installation and checkout phase, an operation and maintenance phase, and sometimes, a retirement phase [3].

Each software product passes through these stages, although the duration, sequence, number of iterations, and exact effect of each stage may vary from product to product. In other words, the life cycle of every software product is different. Some products will spend more years in the conceptual stage. There can be a variety of reasons for this. For example, the idea may not be technically feasible due to some hardware limitations; the product idea may not be economically feasible at that point

in time; or the processing capability of the machines may not be good enough for the product to be commercially viable. Other products will be quickly designed and implemented and will spend more years in the maintenance phase, being modified repeatedly to fix bugs (faults or errors) or to incorporate new features and functionality required by users. Often, after many years of maintenance, a stage will be reached when it will be cost-effective to develop a completely new product rather than to attempt to maintain the current version yet again. So, just like any other commercial product, every software product has a lifetime, starting as somebody's idea, need, or inspiration and ending up being obsolete, unsupported, and unused.

The different SDLC phases may overlap or be performed iteratively or be combined or be omitted depending upon the software development approach (model) used. Many theories and models have been advanced to describe how software goes through these phases (and whether it goes through all the phases).

The most prominent process models include the waterfall model [3], the spiral model [4], the win-win spiral model [5], the incremental model [6–8], the operational model [9], the transformational model [10, 11], the joint application development (JAD) model [12], the evolutionary development model [13], the component assembly model [14], the cleanroom software engineering model [15], and the concurrent development model [16]. A detailed discussion of these models is beyond the scope of this book. It is important to remember, however, that, irrespective of the model chosen, software goes through the different life cycle phases (although some models may not go through all the phases), and the order in which a project moves forward through the different life cycle phases is determined by the process model.

Recent years have seen the emergence of several lightweight or agile process models. Agile proponents claim that the focal aspects of light and agile methods are simplicity and speed. In development work, accordingly, development groups concentrate only on the functions needed immediately, delivering them fast, collecting feedback, and reacting rapidly to business and technology changes. Agile software process models are characterized by the following attributes: incremental, cooperative, straightforward, and adaptive [17]. Incremental refers to small software releases, with rapid development cycles. Cooperative refers to a close customer and developer interaction. Straightforward implies that the method itself is easy to learn and to modify and that it is sufficiently documented. Finally, adaptive refers to the ability to make and react to last-minute changes. These methodologies are mainly suitable for small projects where the team size is small.

Some of the popular agile methods include extreme programming (XP) [18–20], adaptive software development [21], the Crystal family of methodologies [22–24], the dynamic systems development method [25, 26], and feature-driven development [27–29]. Even though most of these models address certain aspects of SCM (for example, the XP programming principles like collective code ownership, continuous integration, small releases, and refactoring, have relations to SCM), none of them addresses the full functionality of SCM. The success of these projects lies in the fact that they are small and have a very small team size. Also, these models and methodologies are still in their infancy and still evolving. So one will have to wait and see how these agile methodologies will address the use of full-fledged CM systems.

SDLC Phases

Software goes through certain phases before, during, and after the development process. We will now examine these phases in detail, discussing when, why, and how these phases are taken care of by the life cycle models. The various phases or steps in the SDLC are listed as follows:

- Project start-up;
- Requirements analysis and requirements specification;
- Systems analysis;
- Systems design (high-level and low-level or detailed design);
- Development/coding and unit testing;
- System/integration testing;
- Acceptance testing;
- Implementation;
- Project windup;
- Maintenance;
- Retirement.

All of these phases will not be present in all projects. Also, all of the activities described in each phase will not be present in many projects.

Depending on the size, nature, and complexity of the project, many of the activities and even some phases might not be present. Also, in many projects, the activities might be performed in an informal manner. For example, small projects will not have a very detailed requirements definition document (RDD) or systems analysis document (SAD). Moreover, in many projects the alpha and/or beta testing phases could be absent.

The phases described next are for a fairly large project; however, depending on the nature of the project and organizational policies, some of these phases might get clustered together, omitted, or practiced under a different name. Keeping this in mind, let's look at the different phases of software development in a little more detail.

Project Start-up

The start-up phase is sort of a curtain raiser for the project. The project team is formed and the project leader identified. The project is organized (i.e., modules are identified, key members are enlisted, and the people who will carry out the support functions such as internal quality assurance and CM are identified). The senior members of the project team sit down together and prepare the project plan to ensure completion of the project within the cost, time, and resource constraints based on the details available.

The main tasks in this phase are the following:

- Studying the project proposal and contract document (if it is a contracted work), estimation work papers, and other documents available;
- Obtaining clarification on matters such as scope, contractual obligations, and client participation in the project, if required;

- Defining the operational process for the project;
- Deciding on the format and standards for documenting the project plan;
- Documenting the project plan per the chosen structure and format;
- Implementing the SCM system.

This phase also sets up the hardware and software environment for the next phase covering the hardware, system software, standards, and guidelines. The main tasks performed in this phase are listed as follows:

- Ensuring that the environment defined in the project plan is still valid for the next phase and, if not, changing it.
- Checking the availability of each resource that is defined as a part of the environment. It is during this time that requests are made to the respective support groups for supplying or arranging the required resources.
- Ensuring that the required hardware and software are in place.
- Testing the environment, if required.
- Obtaining the working space, machines, and other infrastructure requirements for the team members.
- Adding new procedures or modifying current procedures to be followed by the team.

Most organizations will have their own software development standards and guidelines, and if the work is done for a particular client, the project will have to follow the standards of that company. In cases where project standards are not available, they must be developed and finalized during this phase. The standards for various phases include the following:

- The documentation standard for the RDD;
- The documentation standard for the SAD;
- Guidelines for various analysis techniques such as data modeling and process modeling;
- The documentation standard for the high-level design (HLD) document;
- Database and file design standards;
- The documentation standard for test plans and test specifications;
- The documentation standard for user documents;
- The documentation standard for the low-level design (LLD) document;
- The documentation standard for unit test plans and specifications;
- Programming standards;
- The documentation standard for defect logs;
- System test plan standards;
- Test data preparation standards;
- Testing standards.

The SCM system is designed and implemented in this phase. Implementing the SCM system during the initial phase of the project has tremendous advantages as it will help to get the maximum benefit from the system, and the SCM activities can be started from day one. The SCM system implementation involves a number

of activities such as system design, SCM plan preparation, SCM team organization, infrastructure setup, SCM team training, project team training, and system implementation.

One important task of the SCM system implementation is designing the SCM system and preparing the SCM plan. The project team leader and other members, such as representatives from the QA team and the SCM manager, sit together and design the SCM system and SCM procedures to be used in the project. The SCM manager is the person responsible for performing the SCM functions in the project. Sometimes the project leader will serve as the SCM manager or in other cases a person will be designated for that post. If the company has a standard SCM plan, then that plan is tailored for the project. The proposed SCM system is documented in the SCM plan. If the company has a standard SCM plan, then that plan is tailored for the project. The proposed SCM system is documented in the SCM plan.

Each project is different and requires different approaches to SCM. Such factors as the level of formalism, the number of procedures, the change control process, and the SCM organizational structure will vary depending on the size, nature, and complexity of the project. So, for each project the SCM implementation has to be done afresh. The organization may have many projects that already have SCM systems in place with the SCM tools already implemented. Even in those cases, the SCM system should be customized and a separate SCM plan prepared to suit the individual needs of the project. The presence of an established SCM system and tool will make things easier, as all the SCM system designers have to do is to customize the existing procedures, practices, and guidelines for the current project. If the company is using enterprise-wide change management tools, then the tool infrastructure is already in place. However, tasks such as obtaining the tool support for the project and getting the workspaces and libraries allocated will still need to be completed.

The output of this phase, which is usually conducted by a high-level team (including, for example, the project leader, management representatives, the SCM manager, and support team representatives), is the project plan, the SCM plan, and the standards for the next phases.

Requirements Analysis and Requirements Specification

During this phase, user requirements are captured and documented, and a detailed plan for the phase is prepared. The high-level activities for this phase are expanded so that each activity spans not more than one or two person-weeks. The dependencies between the various activities of this module are identified, and the activities are scheduled. Plans for housekeeping activities like backup/recovery and security are formulated. The resources required are estimated, and the team members are allocated tasks.

One of the main tasks of this phase is understanding the current system (manual or computerized). This is not applicable for a new product development project where the task is understanding the functions the software is supposed to perform. The task should be undertaken with a view to examining its adequacy and identifying problem areas. The main tasks that are performed in this phase are understanding

the current system by discussing it with users and studying the documentation available. The main areas to be studied include organization objectives, activities, procedures, rules and standards, files and interfaces.

Every existing system, whether manual or computerized, will have some problems or inadequacies. That is why it is being redesigned. Even if the system is functioning smoothly, there might be areas that could be improved. These existing problems, possibilities, and constraints are identified. The existing system, problems, and constraints are documented for future reference, and the findings are discussed with the client or user.

The next step in this phase is the definition of user requirements. The main activities performed in this phase are diagnosing existing problems and defining user requirements. To do this, the context of the problems has to be understood, the scope of the problems has to be assessed, and the user requirements, application requirements, and information requirements have to be determined.

Once the user requirements have been defined, the next step in this phase is to prepare the RDD. Once the initial draft of the RDD is created, it is given to all the parties of the software project—users, clients, project team, and the support functions. After incorporating suggestions from all quarters, the final RDD is prepared.

The output of this phase is the documentation of the existing system and the RDD. The requirements analysis and the preparation of the RDD are usually done by systems analysts in collaboration with users.

From this phase onward, the major SCM activities—configuration identification, change management and control, status accounting, configuration reviews, and audits—are performed. The different documents (including standards, guidelines, the SCM plan, and the project plan) that are identified as configuration items (CIs) are named; their physical and functional characteristics are recorded; and they are brought under SCM control. The CIs in this phase will contain primarily the updated documents from the previous phase and the RDD. The RDD is usually put under configuration control. So once the final RDD is prepared, it is reviewed and approved, and a baseline is established. Normally, this is the first baseline and consists of the documents from the previous phase and the approved RDD. This baseline is known as the functional or requirements baseline. By establishing a baseline, the functional and other requirements described in the RDD become the explicit point of departure for software development, against which changes can be proposed, evaluated, implemented, and controlled. The requirements baseline usually is the first established baseline in the SCM process.

Systems Analysis

In the systems analysis phase, the proposed system is defined after analyzing various alternatives. The following are the main tasks performed in this phase:

- *Studying the approved RDD.*
- *Generating alternatives (solutions or designs) for the proposed system.* One must access prior knowledge, customize candidate solutions, partition the system, and prototype the system if necessary.

- *Evaluating alternatives*. One must perform impact or cost-benefit analyses for tangible costs (one-time and recurring costs, such as cost of the tools used in the project) and for intangible costs (procedural and personnel-related costs such as costs of training employees on tools to be used in the project).
- *Selecting an alternative*.
- *Determining system requirements with respect to reliability, performance, security, backup and restore capabilities, error recovery, and other quality factors*.
- *Discussing the proposed system with the client*.

In some cases, the project management may decide to develop a prototype of the system to demonstrate an understanding of the user requirements and the functionality that will be provided in the proposed system. Prototyping is required if a lack of clear understanding of the user requirements is considered a major risk in the project. The major activities in prototyping are listed as follows:

- *Determining the objectives of the prototype*. A prototype can be built to demonstrate an understanding of the existing system, functionality of the proposed system, data to be maintained, functions to be provided, data entry screens to be provided, inquiries and reports to be provided, and external interfaces to be provided.
- *Deciding on the type of prototype—that is, whether it should be evolutionary or throwaway*.
- *Deciding on the software and hardware platforms and the tools to be used for developing the prototype and then setting up the environment*.
- *Building a prototype to meet the chosen objectives*.
- *Demonstrating the prototype to the client/users, obtaining feedback, and incorporate suggestions for improvement*.

Once the prototype is developed and the feedback is obtained, the next step is to prepare the SAD, where the proposed system's functionality is documented. While preparing the SAD, a usability plan is also prepared. The usability plan is prepared when the system that is being built uses COTS packages for performing some tasks in the system. The usability plan will compare the available packages and help to identify the one that is best suited for the system in terms of such factors as cost effectiveness, amount of customization, and method of integration of the selected package into the software system. This plan is needed only if the system uses off-the-shelf packages.

During this phase, the project plan, SCM plan, and the RDD are refined and updated based on the project progress and changes in the scope of the project. The output of the systems analysis phase is the prototype (if developed), the SAD, the usability plan, the updated project plan, the SCM plan, and the RDD. All documents except those produced in this phase are already under configuration control. So changes to them can be made only following the formal change management procedures. The documents produced during this phase are also brought under SCM control.

High-Level Design

In this phase, the system design objectives are defined. The following steps are carried out to design the system properly:

- Studying the SAD and ensuring that requirements are understood so that the high-level design documents can be properly written;
- Understanding the features and capabilities of the hardware and software environments in which the proposed system is to be implemented;
- Studying standards and guidelines prepared for the HLD phase;
- Setting design objectives, constraints, and guidelines with respect to such factors as usability, user interface, performance (response time, memory, and throughput), reliability, design directives, and storage.

Sometimes a prototype is developed to demonstrate the user interface design, screens, navigation, and other features of the system. Developing a prototype in the HLD phase is required if the developers want to demonstrate to the user the design features of the system such as system architecture, user interfaces, and system functionality. Sometimes a prototype is developed during the systems analysis phase to demonstrate an understanding of the user requirements and the functionality that will be provided in the proposed system. If such a prototype exists, then this prototype is refined during the HLD phase to demonstrate the user interface design, screens, navigation, and other features of the system.

It is in this phase that the system components such as modules, programs, functions, and routines are identified. The system components are identified hierarchically to the level required. The inputs and outputs of the system are defined. These include menus, screens, navigation, levels of help and help screens, reports, error messages, and the user interface. The programs for each component are identified and classified. The programs can be classified in various ways, including as online/batch, reports, transactions, drivers, functions, and libraries. The performance requirements for each component are established, and the components that can be reused are identified. The following items are produced as a part of this exercise:

- System components list;
- User interface design;
- Programs and the interface definition between programs;
- Screens and report definitions;
- Screen navigation details;
- Help screens and messages.

The next step in this phase is to define the system architecture. The system architecture is established in terms of security, data access, communication, restart/recovery, audit, and user interface. The system architecture deals with issues such as whether the proposed system will be a client/server system, mainframe system, or geographically distributed system; what technology should be used; and how the communications network should be set up. The program dependencies and

interfaces are identified, and the system architecture for each class of programs is finalized and documented.

Another task in this phase is the creation of a first-cut database and finalization of the database/file design. The database/file design is derived from data model or data store identified during analysis. This should include content, access, and organization of the database/files. The contents of each of the tables/files in the database and the access path are defined. The necessary normalization of the database tables is performed to ensure processing efficiency. This step produces the database design document.

The final task in this phase is preparation of the HLD document. The HLD document is prepared as per documentation standards for HLD. The documents that have been prepared thus far, such as the design objectives document, the system architecture document, and the database design document, are used as the input for the HLD document. The system test plan is a part of the HLD document. So while the HLD document is compiled, the system test plan (STP) and system test specification (STS) are also prepared, and they then form part of the HLD document. The preparation of the initial draft of user documents such as user manuals, capabilities manuals, and tutorials are started during this phase. In summary, this phase produces the following documents: the HLD document, the STP and STS, and the initial draft of the user documents.

All of the documents produced in this phase are reviewed and brought under SCM control. If changes are required for any of these items, then the change procedures are initiated and the changes are effected.

Low-Level (LLD) or Detailed Design (DD)

In this phase, the copy libraries, common routines, and program skeletons to be used are finalized. The HLD is analyzed to understand the system architecture, components, programs, and their interfaces. The standards prepared for the LLD phase are studied. The component libraries to be used for each of the programs in the system are identified, as are the common routines and the input and output for these common routines. If program skeletons or templates are to be used for various types of programs, then the scope and contents of such skeletons and templates are decided. The specifications for the component libraries, common routines, and skeletons are written.

The major task in this phase is to write the specification for each program in the system. Writing program specifications is essential for projects involving developments in procedural languages. For each program and reusable routine identified in the system, the program logic is determined; the structure chart is prepared (if necessary); the inputs, outputs, error messages, and help messages are finalized; and the program specification is prepared. As part of the program specification, the unit test specification (UTS) and unit test plan (UTP) are prepared.

The last step of this phase is the preparation of the LLD document consisting of program specifications for all programs, component libraries, skeletons, and templates of the system. All documents, specifications, and program templates produced during this phase are usually subject to configuration control.

At the end of this phase, the project plan is updated, and the RDD, SAD, HLD, STP, STS, and user documents are refined based on the changes and additional information obtained during this phase. These changes are made following the change control procedures because these items are under CM, and hence unauthorized changes are not allowed.

The baseline that is established at the end of the design phase is usually called the allocated or design baseline. The allocated baseline contains the initial approved specifications that form the basis for the software development and testing. The allocated baseline represents the logical progression from the functional baseline and represents the link between the design process and the development process.

Coding and Unit Testing

During this phase the programs, copy libraries, functions, and other program elements are coded (or generated) and tested (unit testing). The main people involved in this phase are developers and programmers, analysts, the QA team, and testers. Among all of the life cycle phases, this is the phase that involves the largest number of people. The SCM team is up and running and will be involved in activities like change management and control, repository management, defect tracking, change request evaluation, and impact analysis. The following are the major activities during this phase:

- Studying the LLD document, test cases, and data;
- Including additional test cases if needed;
- Coding the programs per the program specifications;
- If the evolutionary prototyping approach is followed, refining the prototype to yield the final code;
- Finalizing all error and help messages;
- Conducting unit testing in accordance with the UTS;
- Recording the test results;
- Logging the following unit test errors: errors external to the program (where the error cannot be fixed in the program being tested), errors in LLD and test specifications, errors caused due to the standards adopted, and errors in the reused code;
- Diagnosing and fixing the errors;
- Updating defect logs;
- Initiating corrective action, as applicable (possibly revisiting the earlier phases of SDLC);
- Consolidating test results and findings and recording appropriately.

The output of this phase is the unit-tested programs, all of which—including the source code, test scripts, test data, test results, associated documentation, and change/problem reports—will be under SCM control. To make any changes to those items, a formal change management process has to be followed. Accordingly, the developers will check-out and check-in programs; change requests and problem reports will be initiated; the change management procedures [such as impact

analysis and configuration control board (CCB) meetings] will be at their peak; and project managers and leaders will be using the status accounting information to ensure that project is on track and under control. The proper operation of the SCM system during this phase of the SDLC is of paramount importance for the success of the project.

System Testing

This is the phase in the SDLC where system testing or integration testing is carried out. System testing is done using STP, STS, and system test data. Many companies do alpha and/or beta testing also.

Alpha testing is done when the system or product has a lot of new previously untested features. Because a lot of the functionality is untested, the development team might be uncomfortable proceeding with the final testing and release of the product until they get a feedback from a limited number of users or customers. So the developers use the alpha testing primarily to evaluate the success or failure (or acceptance) of the new features incorporated into the system.

Beta testing is required when the development team decides that some level of customer evaluation is needed prior to the final release of the product. In the case of beta testing, the developers are no longer looking for user inputs on functionality or features. The product has all the functionality incorporated in it, so the development team will be looking for the beta testers to uncover bugs and faults in the system. Unlike alpha testing the beta testing is done on a much larger scale; that is, the number of people who will be doing the beta testing will be much higher than that for alpha testing. Companies usually distribute the beta releases cost-free to the people who have enrolled in the beta testing program, and in many cases, the beta versions will be available for download from the company's Web site. New products will have the alpha testing followed by the beta testing. In the case of new versions of existing products, however, either alpha or beta testing is done.

The tasks in this phase are as follows:

- Carrying out system test according to the STP and STS. For alpha and beta testing, there will be no test plans. In the case of alpha testing, the testers will be evaluating the acceptability of the new features or functionality; in the case of beta testing, the testers will be trying to find bugs or problems in the product.
- Recording the test results.
- Logging the test errors.
- Diagnosing and fixing errors.
- Updating defect logs.
- Initiating corrective action as applicable. This might involve revisiting earlier phases of the SDLC.
- Performing regression testing.
- Consolidating and reporting test results and findings.

The major players involved in this stage are the QA team, the testers, the development team (for bug or problem fixing), and the actual users of the system. If alpha

beta testing is used, then the number of people who will be testing the system will increase dramatically.

During this phase, members of the SCM team will have their hands full, because they are the ones who coordinate the change requests and problem reports and ensure that the changes are made according to the procedures and that all people concerned are aware of the changes. Once the project is successfully tested, functional and physical configuration audits are performed to ensure that the final product is complete and satisfies the specifications. At this stage, the product baseline is established. The product baseline represents the technical and support documentation established after successful completion of the functional configuration audit and physical configuration audit.

Acceptance Testing

Acceptance testing is the formal testing that is conducted (usually by the user, client, or an authorized entity) to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. This phase is carried out only if the system is developed for a particular client or customer. In this phase, the project team prepares for the acceptance test by ensuring the availability and completeness of all work items needed for the acceptance test and populating the acceptance test data. The project team will assist the client or customer in acceptance testing, recording the errors found, and fixing them. The main tasks in this phase are the following:

- Providing support to the client in conducting acceptance test;
- Ensuring that documentation-related tests are also completed;
- Recording acceptance test results;
- Logging acceptance test errors;
- Diagnosing and fixing errors;
- Updating the defect logs;
- Revisiting earlier phases of the SDLC, as required, in order to fix errors;
- Performing regression testing;
- Preparing a report summarizing the test results;
- Highlighting any disagreements.

Implementation

Once the integration, system and acceptance (in some cases, alpha and beta testing) testing are over, the software product or the system is turned over to the customers or clients or installed at the client site. The members of the development team will supervise the installation in the case of large projects. In the case of a shrink-wrapped project, the customer does the installation and, if faced with any problem, calls the technical support team of the vendor. In the case of large projects, the installation team, in collaboration with the end users, will install the system and train the users to operate the system. There will be some amount of user training. Once the system is up and running, and the users are familiarized with the product, the implementation phase is complete. From that point onward, the maintenance team

will take over and manage tasks such as technical support, product enhancements, and error fixing.

Project Windup

In this phase, the project windup activities are completed, and all the resources acquired for the project are released. The main activities are listed as follows:

- Carrying out project-end appraisals;
- Releasing project team members and hardware and software resources;
- Returning client-supplied products, if any;
- Ensuring availability of project documentation copies in the library.

Project Maintenance

Once the system has been developed and tested, it is released to the users. From this point onward, the maintenance phase starts. Once people start using the system, many errors that escaped the testing will be found. Users might ask for new features and enhancements. It is the responsibility of the maintenance team to attend to these requests and to fix the bugs that are found.

It is during the project maintenance stage that the full impact and usefulness of the SCM process can be felt. If the project was developed following a good SCM system, then all the resources such as defect and defect-prevention details, help desks, programs, libraries, change histories, technical documents, and user documents will be readily available. Furthermore, since any particular component or version can be recreated with very high accuracy, the change-management and problem-solving processes—and the overall tasks of maintenance and technical support—will be much easier.

Retirement

The final phase in the life cycle of a software product or system is retirement. After many years of service, software products or systems reach a stage when any further maintenance would not be cost-effective. This might happen when the proposed changes are so drastic that the whole design has to be changed, and as a result, it would be cheaper to redesign and recode the entire product from scratch.

Similarly, changes made to the original design may have inadvertently built interdependencies into the product, creating a real danger that even a small change to one module will have a drastic effect on the functionality of the product as a whole. Yet another reason for retirement is that the documentation may not have been adequately maintained, thus increasing the risk of a regression fault to the point where it would be safer to recode than to maintain. Another reason for retiring a software product is that technological advancements have made the existing system obsolete. In such cases, the hardware on which the product runs has to be replaced by a different (more powerful and less expensive) machine with a different operating system, and it is cheaper to rewrite from scratch than to modify the

product. In each of these instances, the current product is retired, and a new version will be developed and the life cycle continues.

During this phase, the SCM system that was designed for the particular project and documented in the SCM plan is also retired. However, the SCM tools used are not retired; they are released and used by other projects. Similarly, the SCM team members and the maintenance and support team members are released to other projects. Likewise, records and documentation created during the project with any value—either due to legal reasons or as reference material—are retained per the records-retention policy specified in the SCM plan. Usually they are archived, and unwanted documentation and records are discarded.

True retirement (removal of a product) is a rare event that occurs when a product has outgrown its usefulness. The client organization no longer requires the

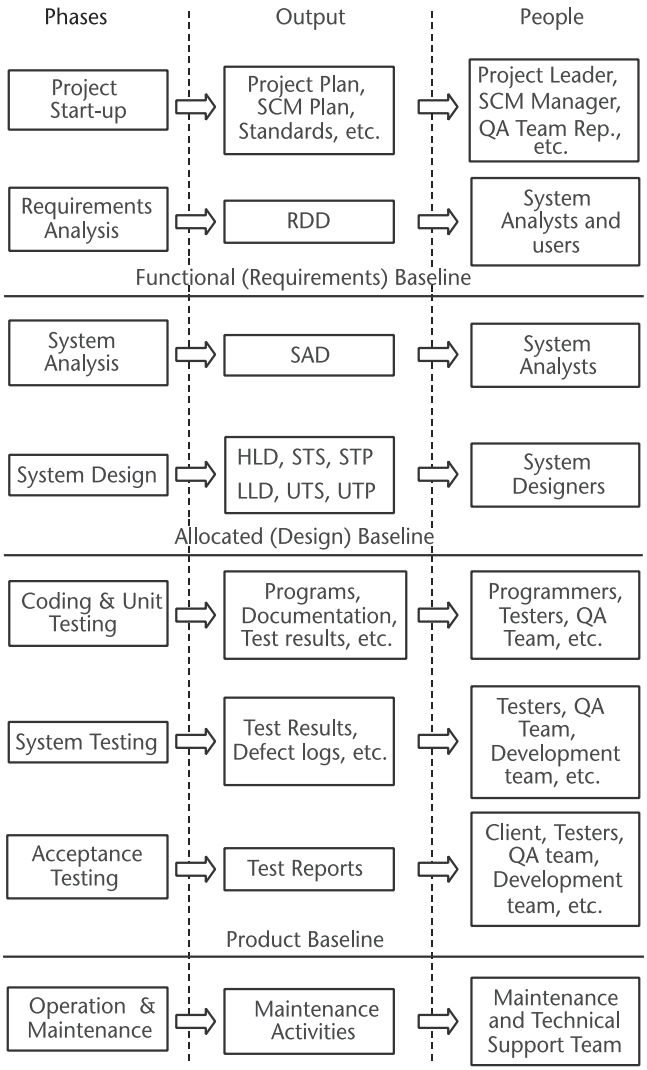


Figure 2.1 Life cycle phases and their relationship with SCM.

functionality provided by the product, and it is finally removed from the computer on which it has been in operations mode for many years.

Summary

This chapter describes the various phases involved in the software development process. All of these phases may not be present in all projects, and the order in which the various steps are executed will vary.

Most of the time, some degree of overlap will occur between the various phases. The software life cycle model that is adopted will determine these things. The chapter describes the various outputs of each phase and the role of SCM in the project life cycle and discusses how and when the different SCM activities are performed in a project. These aspects are summarized in Figure 2.1.

References

[1] Humphrey, W. S., *Managing the Software Process*, New York: Addison-Wesley, 1989.

[2] Jones, G. W., *Software Engineering*, New York: John Wiley & Sons, 1990.

[3] Royce, W., "Managing the Development of Large Software Systems: Concepts and Techniques," *Proc. IEEE WESCON*, 1970.

[4] Boehm, B. W., "A Spiral Model for Software Development and Enhancement," *IEEE Computer*, Vol. 21, No. 5, 1988, pp. 61–72.

[5] Boehm, B. W., et al., "Using the Win-Win Spiral Model: A Case Study," *IEEE Computer*, Vol. 31, No. 7, 1988, pp. 33–44.

[6] McDermid, J. A., and P. Rook, "Software Development Process Models," in *Software Engineer's Reference Book* (J. A. McDermid, ed.), Boca Raton, FL: CRC Press, 1994, pp. 15.26–15.28.

[7] Brooks, F. P., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 20, No. 4, 1987, pp. 10–19.

[8] Mill, H. D., "Top-Down Programming in Large Systems," in *Debugging Techniques in Large Systems* (R. Rustin, ed.), Englewood Cliffs, NJ: Prentice-Hall, 1971.

[9] Zave, P., "The Operational Versus Conventional Approach to Software Development," *Commun. ACM*, Vol. 27, No. 2, 1992, pp. 104–118.

[10] Balzer, R., "Transformational Implementation: An Example," *IEEE Transactions on Software Engineering*, Vol. 7, No. 1, 1981, pp. 3–14.

[11] Balzer, R., "A 15-Year Perspective on Automatic Programming," *IEEE Transactions on Software Engineering*, Vol. 11, No. 11, 1985, pp. 1257–1268.

[12] Wood, J., and Silver, D., *Joint Application Development*, New York: John Wiley & Sons, Inc., 1995.

[13] Lehman, M. M., and L. Belady, *Program Evolution: Processes of Software Change*, London: Academic Press, 1985.

[14] Nierstrasz, O., "Component-Oriented Software Development," *Commun. ACM*, Vol. 35, No. 9, 1992, pp. 160–165.

[15] Dyer, M., *The Cleanroom Approach to Quality Software Development*, New York: John Wiley & Sons, 1992.

[16] Aoyama, M., "Concurrent Development of Software Systems: A New Development Paradigm," *ACM SIGSOFT Software Engineering Notes*, Vol. 12, No. 4, 1987, pp. 20–24.

[17] Abraharnsson, P., et al., *Agile software development methods: Review and Analysis*, Espoo, Finland: Technical Research Centre of Finland, VTT Publications, 2002.

[18] Beck, K., *Extreme Programming Explained: Embrace Change*, Boston, MA: Addison-Wesley, 2000.

[19] Beck, K., and M. Fowler, *Planning Extreme Programming*, Boston, MA: Addison-Wesley, 2000.

[20] Jeffries, R., A. Anderson, and C. Hendrickson, *Extreme Programming Installed*, Boston, MA: Addison-Wesley, 2000.

[21] Highsmith, J. A., *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, New York: Dorset House Publishing,2000.

[22] Cockburn, A., *Surviving Object-Oriented Projects: A Manager's Guide*, Vol.5, Boston, MA: Addison Wesley Longman, 1998.

[23] Cockburn, A., *Writing Effective Use Cases: The Crystal Collection for Software Professionals*, Boston, MA: Addison-Wesley Professional, 2000.

[24] Cockburn, A., *Agile Software Development*, Boston, MA: Addison-Wesley, 2002.

[25] DSDM Consortium, *Dynamic Systems Development Method (Version 3)*, Ashford, England: DSDM Consortium, 1997.

[26] Stapleton, J., *Dynamic Systems Development Method: The Method in Practice*, Boston, MA: Addison Wesley, 1997.

[27] Palmer, S. R., and J. M. Felsing, *A Practical Guide to Feature-Driven Development*, Englewood Cliffs, NJ: Prentice Hall PTR, 2002.

[28] Rising, L., and N. S. Janoff, "The Scrum Software Development Process for Small Teams," *IEEE Software*, Vol. x, No. x, 2000, pp. 2–8.

[29] Schwaber, K., and M. Beedle, *Agile Software Development with SCRUM*, Englewood Cliffs, NJ: Prentice Hall, 2001.

Selected Bibliography

Behforooz, A., and F.J. Hudson, *Software Engineering Fundamentals*, New York: Oxford University Press, Inc., 1996.

Jones, G. W., *Software Engineering*, New York: John Wiley & Sons, 1990.

Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

Peters, J. F., and W. Pedrycz, *Software Engineering: An Engineering Approach*, New York: John Wiley & Sons, Inc., 2000.

Pfleeger, S. H., *Software Engineering: Theory and Practice (2nd Edition)*, Pearson Education, Inc., 2001.

Pressman, R. S., *A Manager's Guide to Software Engineering*, New York: McGraw-Hill, 1993.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (5th Edition)*, New York: McGraw-Hill, 2001.

Schach, S. R., *Software Engineering*, Boston, MA: Aksen Associates, Inc., 1990.

Shooman, M. L., *Software Engineering: Design, Reliability and Management*, New York: McGraw-Hill, 1983.

Sommerville, I., *Software Engineering (6th Edition)*, Pearson Education Ltd., 2001.

CHAPTER 3

Pitfalls in the Software Development Process

Introduction

The software development process is very different from other production or manufacturing processes. According to Jones [1], software products are intangible, because there is no need for physical mechanisms, structures, or processes. Software engineers do not use most of the concepts familiar to traditional engineering, and their work is mostly independent of natural science. Also, software products are much more complex and sophisticated, thus requiring special care in conceptualizing, managing, organizing, and testing. Software products are manufactured by a simple copying process, so almost all of the production effort is dedicated to design and development.

Because the software development process is different from regular industrial practice, normal rules of production or manufacturing do not apply here. For example, Brook's law [2] states that, "Adding manpower to a late software project makes it later." It might sound strange, but it is true. If you calculate the time required to train the new people, the added communications channels because of the new people, and other related complexities, the project in question (the already late project) could be finished earlier if more people were not inducted. This is not true in a construction project where additional manpower can speed up the project.

So now that we have established that the software development process is different from the other manufacturing or production processes, we need to look at some of the problems that plague many software projects and that can result in time and cost overruns if corrective actions are not taken. The most frequent among these are the communications breakdown problem, the shared data problem, the multiple maintenance problem, and the simultaneous update problem. This chapter examines each of these problems in greater detail, since understanding them is crucial to realizing the importance of the CM function.

Communications Breakdown Problem

The era when a single person developed a software product is long gone. Today's software projects consist of teams with hundreds of members in different modules. The modules or subsystems of a project might be located in different continents and

might very well include developers with different social, cultural, and educational backgrounds.

In a single-person project, communications breakdowns never occur. According to Rawlings [3], "When only one person is working on a project, that one person has a rather singular communication path with no need for interpretative cognition. The person has only him or herself to communicate with and, hopefully understands his or her own thought processes. When two people are working on the same project, there are now two communicators and two listeners with four potential communication paths. Not only is there a dramatic increase in the number

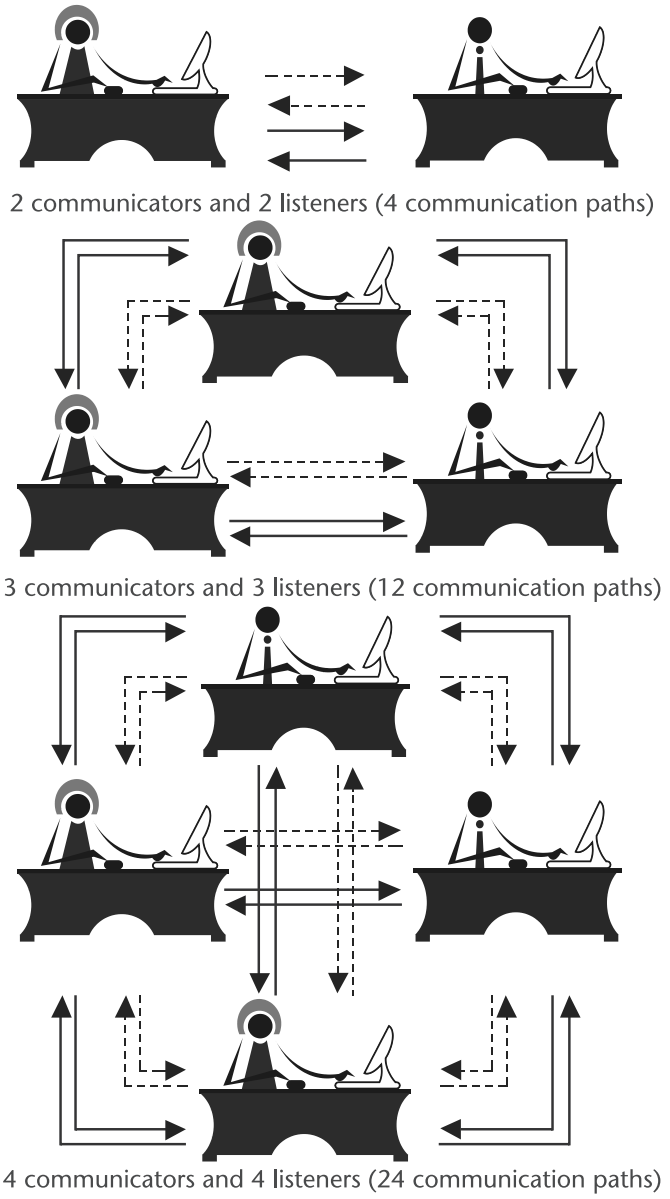


Figure 3.1 Increase in number of communication paths with increasing team size.

of communication paths, there is also the problem of interpretative cognition, which now comes into play."

As more and more people are added to the project team, the total number of communication paths increases dramatically as shown in Figure 3.1, and as the number of communication paths increases, the potential for communication errors also increases.

Interpretative cognition is a part of the process that occurs when two or more people communicate with one another. It is a measure of how much of a person's communication is understood by the other person or persons. You must have encountered situations in daily life when you said something and the listener understood something else, resulting in misunderstanding and confusion. In such cases we say that the interpretative cognition did not work. When a person wants to communicate some idea, he or she must describe it using such tools as words, pictures, drawings, and gestures. The person who is listening to this communication should see and hear the communication directed at him or her and should reconstruct the idea in his or her mind. If the two ideas are the same, then we say that the communication has been successful.

The complexity of this process increases as the idea that is communicated becomes more complex or if the people who are involved are not familiar with one another's mannerisms and communication methods. Failing to understand a gesture or body language can convey the wrong meaning to the listener.

So in the case of large software projects, where complex and sophisticated systems are being developed, the ideas that are to be communicated are complex. Also, members of the project team, as mentioned previously, may be working in different parts of the world; thus, the chance to communicate face to face will be rare. Moreover, because of the different cultural and ethnic backgrounds, the gestures, phrases, and colloquial usages will not be understood by all team members. So the lack of proper communication between team members or a communications breakdown can result in the failure of the project.

How do you control communications and keep everyone informed about the tasks and activities that affect them? How will a project leader make sure that all team members are communicating with each other and that all people are aware of what is happening? How will he or she ensure that the effort is not duplicated and that the work of one person is not destroyed by another?

In the case of small projects, where the number of people involved is limited to two or three, effective communication is easier to establish. If the team members have been working together for quite some time and are familiar with each other's communication patterns and work methods, then the chances of a communications breakdown and associated problems can be minimized, but not completely eliminated. Even in projects involving two or three people, efforts can be duplicated; one can overwrite the code the other person has just fixed, and so on.

If in a small project, the problems just mentioned can happen, then think about a project having, say, five modules and 100 members. It will be total chaos if some sort of control mechanism is not in place and that control mechanism is CM.

SCM helps prevent the communications breakdown by controlling and managing change. SCM ensures that if something is changed, then all the people who need to know about the change are made aware of it. Configuration status accounting (CSA)

is an SCM function that captures all the activities in a project and keeps accurate and traceable records of these activities. CSA produces reports relating to various aspects of the software development process like items changed, who changed them, why they were changed, and so on. The SCM database can also be queried using query languages for more specific information that is not available in CSA reports. All these go a long way in avoiding the communications breakdown problem.

Shared Data Problem

The shared data problem is a very common source of trouble in any environment where two or more programmers or programs share a common resource. It can be a function that is shared by two programmers. It can be a component library that is common to two programs. It can be a housekeeping program or an error-handling subroutine that is being used by all the programs in the project. The trouble arises when one developer makes a change to any of the common or shared resources without telling others.

Consider two programmers, Bob and Jane, sharing a function (Figure 3.2). To improve functionality, Bob makes some changes to the function. Jane is not aware of the change. However, the next time Jane tries to execute Bob's program it may "abend," or it may not function correctly depending on what changes Bob has made. Jane is completely in the dark about the change made by Bob to the function. She is amazed by the fact that the program that was working fine up to that point is suddenly not working. She can spend hours in debugging, but without some luck, she will never find the cause, as she has no reason to doubt the function, which is the real culprit. This type of situation occurs often in almost all projects where more than one person is involved and, in some cases, in single-person projects also.

The software developers found a way to solve this problem: by creating separate and independent workspaces where each programmer has his or her own copy of the resources he or she needs (Figure 3.3). In this situation, even if one programmer modifies the code of a shared resource, others are not affected because they are using separate copies of the same resource. This solution has one major drawback, however: It creates multiple copies of the same function or program throughout

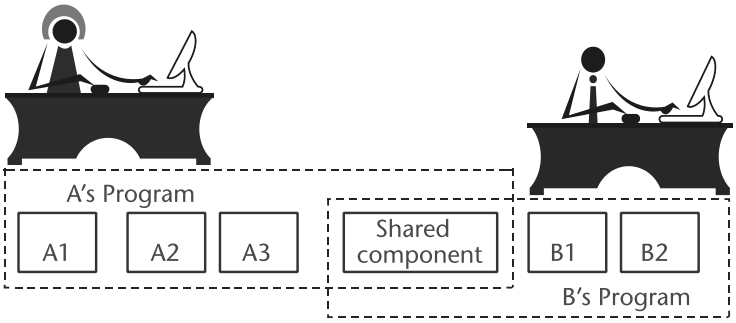


Figure 3.2 Shared data problem.

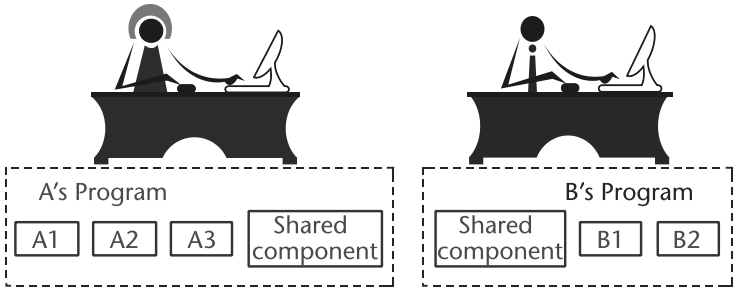


Figure 3.3 Shared data problem solved by using independent workspaces.

the project, which, in most cases, will not be identical. As a result, a lot of space is wasted. The real trouble, however, is that this creates another problem: the multiple maintenance problem.

Multiple Maintenance Problem

This is a variation of the shared data problem. It occurs when there are multiple copies of the shared components in the system. The main problem created by having multiple copies is keeping track of them (Figure 3.4). How many copies of the function exist in the project? Which program uses which copy? How many copies are still in the original state? What changes were made? To which copies were those changes made? Ideally, all the copies across the system would be identical, but rarely is the situation ideal.

Suppose Bob finds a bug and makes the necessary corrections to a function. It is his duty to inform all the people who are using a copy of the function in question that he has made the change; otherwise, while he will be using the "bug-fixed" version of the function, everyone else will be using the one with the bug. When the time comes to integrate all of the common functions, the multiple copies will create problems for the programmers depending on which version of the function is actually used. So, if the new version—the one that is fixed—is used, then all the programs that have used the old version will be in trouble.

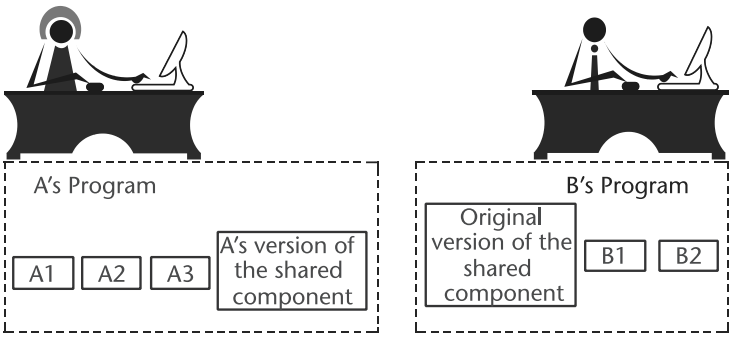


Figure 3.4 Multiple versions of the same component in use.

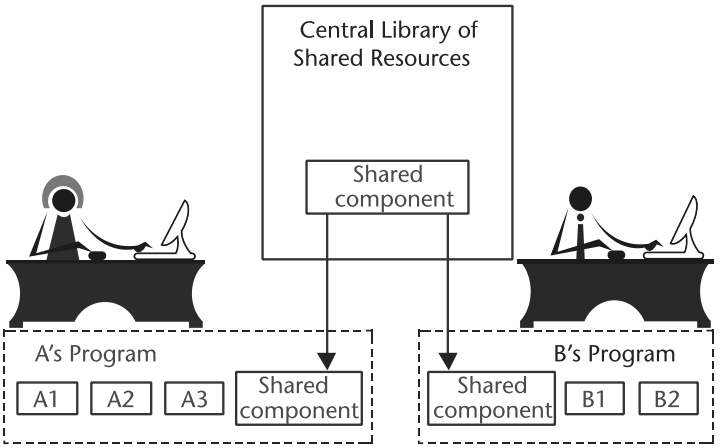


Figure 3.5 Solving the multiple maintenance problem using a central library.

As with any problem, programmers found a way to solve the multiple maintenance problem, too. They created centralized libraries (Figure 3.5).

In this solution, the shared components were kept in a central location. The different programmers took the required resources from the central library. If a bug was discovered in a function, then the copy in the library was updated, so that it was available to all. These centralized libraries could be considered the predecessors of today's repositories.

There was a problem, however, with these shared libraries. There was no control over the changes made to a shared resource (e.g., program, function, or specification) in the central library of shared components. Anybody could make a modification. There was also no formal mechanism for informing all users that a particular module or function had been changed. Further, there was no proper authority to decide whether a particular change was necessary or not. This lack of proper control mechanism led to another problem—the simultaneous update problem.

Simultaneous Update Problem

Consider this situation. Bob has found a bug and has fixed it. He copies the bug-fixed version to the central library, thus overwriting the existing copy. According to the procedure we saw in the previous section, this is how it should be done. That is, the changes are incorporated to the copy in the central library where all the shared resources were kept.

But what if Jane also finds the same bug? She is not aware of the fact that Bob has found the bug and is fixing it or has already fixed it. She also fixes the bug and then copies the function to the library, thus overwriting the copy that was created by Bob. So the work that was done by Bob to fix the bug is lost.

This creates a lot of problems (Figure 3.6). One, a lot of time and resources are wasted because two people, working in isolation, corrected the same mistake. Two,

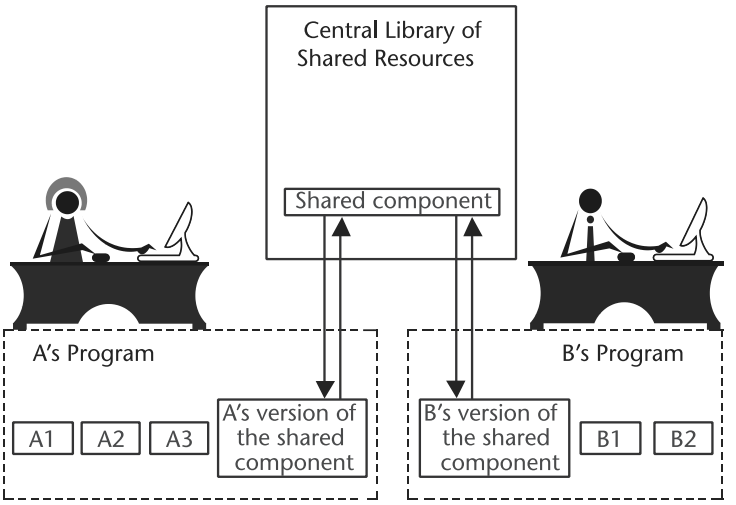


Figure 3.6 Simultaneous update problem.

different people will have different opinions about how to fix the same problem. Who will decide which one is the best and which one is to be used?

Now consider another variation of this problem—and one that is much more serious. Bob finds a bug and fixes it. During the same time, Jane finds a different bug and fixes it. However, depending on which programmer updates the library copy last, the other programmer's work is lost. Still, both bug fixes are necessary and need to be incorporated into the function. This means that the uncontrolled and unmanaged functioning of these central libraries where the shared resources are kept is not going to solve the problem. In other words, just by creating a repository of shared components and leaving it unmanaged and uncontrolled so that anybody can make changes to the items in the repository is not going to solve the problem.

SCM solves the three problems mentioned above—the shared data problem, the multiple maintenance problem, and the simultaneous update problem—by enforcing formal change management and control procedures. Once an item is brought under SCM control, it is stored in a controlled library where the access is restricted. Developers just cannot make changes to items—configuration items—in the controlled library. A change request has to be initiated, impact analysis has to be done, and the change has to be approved. Only when the change is approved will the item be released from the controlled library. Once the change has been made and tested, the item(s) have to be reviewed and approved before being returned to the controlled library. Since the developers cannot make changes to programs at will, the shared data problem is eliminated. Since there is only one library to keep the configuration items, and since only one person[1] can make changes to an item at a time, the multiple maintenance problem is also eliminated. Since the access to the central library is controlled and formal change control procedures have to be followed to modify the items, the simultaneous problem is also eliminated.

1. There are instances where more than one person needs to make changes to the same item and SCM has ways to control such situations—branching and merging (detailed in Chapter 5).

Summary

We have considered the four major problems that create trouble in the software development process—and drive programmers and developers crazy in the process. Fortunately, a good change management and control system can solve these problems and can bring discipline into the development process and improve the development productivity. A lot of time that would otherwise be spent on debugging and reworking can be saved. It is a well-known fact that the earlier defects are found, the easier and cheaper it is to solve them. In order to detect and solve problems and to ensure that the solved problems do not recur, it is important to have a good SCM system from the beginning of the project.

References

[1] Jones, G. W., *Software Engineering*, New York: John Wiley & Sons, 1990.

[2] Brooks, F. P., *The Mythical Man-Month*, New York: Addison Wesley Longman, 1995.

[3] Rawlings, J. H., *SCM for Network Development Environments*, New York: McGraw-Hill, 1994.

CHAPTER 4

Need and Importance of SCM

Introduction

Chapter 1 introduces SCM; Chapter 2 provides a brief introduction to the software development process and how the various SCM activities fit into the development process. Chapter 3 discusses some of the most common problems that plague the software development process. This chapter examines why SCM is important and why it should be implemented in all software projects irrespective of the size (small, medium, large, or very large); complexity (simple or complex); and the stage (conceptual, design, development, testing, or maintenance) of the project.

Need for SCM

Here are just some of the reasons for implementing SCM:

- The nature of software products, projects, and development teams;
- The increased complexity of the software systems;
- The increased demand for software;
- The changing nature of software and the need for change management.

The Nature of Software Products, Projects, and Development Teams

Software products have many different components in multiple versions and run on many different software and hardware platforms. The versions and variants are closely interrelated, and if no control mechanism is in place, they are easily modified, corrupted, or destroyed.

Software development projects are one of the most difficult types to manage. In a software project, everything from requirements to the development and installation environment is constantly changing: Users modify the requirements they had laid out at the beginning of the project; programmers make changes to code without letting other users of the same code know about those changes; development environments change due to new developments in technology; installation and operational environments change because of customer policy decisions or technological innovations or obsolescence; end users differ in their knowledge of the system they are going to use—the list of the difficulties in a software project is endless. To manage such a complex activity, project managers need tools to avoid chaos and confusion, some examples of which are listed as follows:

- Tools that will ensure that changes happen in an orderly and scientific manner;
- Tools that prevent communications breakdown among project stakeholders;
- Tools that alert managers before something goes wrong (an early warning system);
- Tools that record everything so that when disaster strikes troubleshooting is easier.

Today, most software products are developed by teams spread across the world and by people who have different social, educational, cultural, and ethnic backgrounds. Project managers need a system to ensure that these teams can communicate smoothly without misunderstandings. Employee turnover is another factor that troubles all projects. When an employee leaves a project, it leaves a hole, and it takes a long time to recover from the loss as some portion of the project-specific knowledge is lost. To prevent this problem, projects need a system to record all the details of the work that has been done and that needs to be done, including, for example, what the employee did, how he or she did it, what tasks are left unfinished, and which other components are affected. Then, even if an employee leaves the project, somebody else can take his or her place and get up to speed much more quickly.

So it is clear that the special nature of the software products, projects, and teams make software development very unique and very complex. Attempting to succeed in such a venture without any tools or systems (such as SCM) is like attempting a tightrope walk without any safety precautions. The most experienced and successful adventure sports personnel are the ones that do not make any compromises on taking safety measures and precautions. Similarly, the more experienced and successful software project managers insist on scientific methods and tools for managing the project. Experience has taught them that the difference between a successful and failed project is the use of proper control mechanisms and scientific procedures.

Increased Complexity and Demand

Information technology is revolutionizing the way in which we live and work. It is changing all aspects of our lives and lifestyles. The digital revolution has given humankind the ability to treat information with mathematical precision, to transmit it with a high degree of accuracy, and to manipulate it at will. These capabilities are bringing into being a whole world within and around the physical world.

The amount of information-processing power that is available to humankind is increasing. Computers and communications are becoming integral parts of our lives. The driving force behind these advancements is computer software. Computer software is becoming more and more complex, and the amount of software being developed each year is increasing at an exponential rate. Also, software is being used to control a range of activities from mission-critical applications such as controlling the operations of satellites and intercontinental ballistic missiles, managing the functioning of banks and hospitals, and handling airline and railway reservation systems to performing mundane tasks like operating door-locking systems or desktop publishing.

Musa [1] estimates that the demand for software systems increases by 900% each decade. Boehm and Papaccio [2] predict that the expenditure on software

development increases by 200% each decade, whereas the productivity of software professionals increases by only 35%. So the gap between supply and demand is very large. Software companies, nonsoftware organizations, and governmental agencies are producing hundreds of new applications and modifying thousands of existing ones every year. All of them are finding it difficult to develop the high-quality software they need on-time and within budget.

Another aspect of software that has changed is the complexity. According to Jones [3], in the early days of software development, computer programs were typically less than 1,000 machine instructions in size, required only one programmer to write, seldom took more than a month to complete, and often shouldered development costs of less than $5,000. Today, however, some of the large systems exceed 25 million source code statements, usually require thousands of programmers, can take more than five years to complete, and have development costs in the range of $500 million.

In the early days of software development, all of the parts or modules of a software system were developed in the same place. Today, however, the different components of complex software systems are not even built by the same organization. Many software systems are built jointly by different organizations working in different parts of the world. They may communicate via Internet, e-mail, or videoconferencing technologies. So in this distributed development environment where face-to-face communication is rare, managing and coordinating the development process is a difficult task.

This increasing demand for new software, the need to modify or maintain existing software, the increasing complexity of the software development process, and the critical nature of the applications in which software is being used dictate that software development cannot be accomplished in the same way it was during the early days. As Jones [4] has stated, "Software has become the central component in many complex activities. For this reason, the challenge of producing it requires specialized and powerful techniques. It is not possible to rely on luck, guesswork, and innate talent for dependable results." We need scientific methods and techniques for developing software.

The Changing Nature of Software and The Need for Change Management

Software systems are subject to constant changes—during design, during development, and even after development. The pioneering work in this area has been done by Lehman and Belady [5] and is detailed as a set of laws called Lehman's laws. According to Lehman's law of continuing change, any large software system that is being used will undergo continual change, because the system's use will suggest additional functionality. It will change until it becomes more cost-effective to rewrite it from scratch [6]. This means that the software will be subject to constant changes other than the bug fixes and defects that are already in the software and that will be detected during and after its development.

That's not all: The software system that is perfectly developed and that has met all requirements and passed all audits and reviews will also change. According to Lehman [7], even if a system were built in complete conformance to the requirements,

the system would still evolve because the system is introduced into the real world, and the environment into which the system is introduced is subject to change. So in order to adapt to the changes in the environment in which the system works, it has to change. In other words, no matter how perfectly built a system is, it will have to be changed to meet the changes in the environment. So it is clear that the only constant thing about software is change. If changes are not managed, chaos and confusion will result. Accordingly, a mechanism for managing the change and controlling it is required. SCM is one such mechanism.

Benefits of SCM

We have seen the need for SCM. A well-designed and properly implemented SCM system offers many benefits to developers, organizations, and customers. Some of the benefits of SCM are listed as follows:

- Improved organizational competitiveness;
- Better customer service and improved customer goodwill;
- Better return on investment;
- Improved management control over software development activities;
- Improved software development productivity;
- Easier handling of software complexity;
- Improved security;
- Higher software reuse;
- Lower software maintenance costs;
- Better quality assurance;
- Reduction of defects and bugs;
- Faster problem identification and bug fixes;
- Process-dependent development rather than person-dependent development;
- Assurance that the correct system was built.

These benefits are discussed in the following sections.

Improved Organizational Competitiveness

Today's business environment is highly competitive. Organizations are fiercely competing for market share. In this race to survive, only the organizations that continuously improve their processes and implement and use scientific tools, techniques, and business practices will succeed. Software development is a process that has many pitfalls (as seen in Chapter 3) and many peculiarities as mentioned in the beginning of this chapter. If not managed properly, the software development process can spiral out of control and result in time and cost overruns and low-quality products. SCM is the discipline of controlling the chaos of the software development process by managing uncontrolled change and improving communication. SCM can improve the productivity of employees by reducing confusion, redundant work, and wasted effort. It can reduce the time to market and help organizations produce high-quality products. Thus, organizations can use SCM as a weapon to

increase their competitiveness. The difference between the market leader and an "also ran" is often the presence of a good SCM system and an organizational culture that promotes effective use of the SCM system.

Better Customer Service and Improved Customer Goodwill

Even though it is possible to drastically reduce the number of defects in a software product, it is impossible to completely eliminate all defects. Even if one manages to deliver a completely defect-free product to the customers, there will be requests for enhancements. So once the product is released into the market, the maintenance and technical support personnel should gear up for such tasks as solving technical problems, incorporating enhancement requests, and fixing bugs. To make changes to the software (for incorporating new features or fixing bugs), the support team must identify the source of the bug or the place where the enhancement is to be made. This process is a lot easier if the organization has implemented a good SCM system from the very beginning and followed the SCM procedures. Since the SCM system keeps a record of all the changes made to each and every CI, identifying the cause of a problem is easy. Since the SCM help desk contains a record of all known problems with the system and their solutions, troubleshooting is easier. Thus, a good SCM system leads to quicker resolution of customer problems, thereby improving customer goodwill.

Better Return on Investment

A good SCM system automates the process of changing and deploying software applications. The SCM system helps in process improvement, process automation, communication, coordination, and change management and provides a better return on investment (ROI) by accomplishing the following:

- Reducing rework, wasted effort, and number of errors;
- Shortening development and change cycles and reducing the time to market;
- Improving the quality of products and reducing errors;
- Reducing the time to find and fix errors and bugs;
- Automating and streamlining build and release management;
- Increasing productivity for human resources throughout the organization.

Improved Management Control Over Software Development Activities

SCM systems improve communication in software development projects. The status accounting function provides the project and company management with up-to-the-minute information about what is happening in the project so that management can take proactive actions to keep the project on-time and within schedule. The status accounting information also includes information about the quality of the software development (as indicated by such factors as the number of changes and problems) and process improvement (indicated by metrics like average time to fix bugs and time spent on development), which will help management to measure the status of the project and manage it better.

Improved Software Development Productivity

Chapter 3 examined the various problems—communications breakdown problem, shared data problem, multiple maintenance problem, and simultaneous update problem—that reduce the productivity of software professionals and result in wasted effort, duplicated effort, and a host of other complications.

If software development were carried out in an environment where these problems did not occur, then productivity would naturally increase, because problems and mistakes would be reduced. For example, if the communication channels are well-defined and functioning smoothly, if changes are made in a controlled fashion, if all team members are aware of how to handle change (i.e., each member of the team knows what to do when they have to change something), then a lot of time and effort can be saved. With an SCM system in place, developers have more time to develop software instead of wasting time looking for missing items, fixing bugs that have already been fixed, and solving problems caused by using different versions of the same code. This yields a tremendous improvement in development productivity.

Easier Handling of Software Complexity

We have seen that software development is a very complex process. The use of SCM in a software development project will equip all the project stakeholders to better handle the complexity. SCM identifies all the components or artifacts of a software project, captures their description and physical and functional characteristics, and scientifically and systematically classifies, categorizes, and names them. These components or CIs are then stored in one or more protected environments so that they are safe from unauthorized changes, corruptions, and even disasters. SCM tracks the progress of each component and alerts the management of impending dangers, so that preventive actions can be taken. Thus, SCM acts as a management-reporting tool that tells the project manager whether the project is on schedule. SCM also performs audits and reviews to ensure that the product that is being developed is exactly the same as the user requires (as specified in the requirements document). Thus, SCM prevents chaos, confusion, rework, and wasted effort in a software development project, helping to make the complex task of software development a lot easier.

Improved Security

We have seen that SCM prevents unauthorized changes to the different components of the project. SCM achieves this by placing the reviewed, approved, and baselined CIs in controlled libraries where access is restricted. SCM also maintains a change log (history of the evolution of the CI to its current state). Such change logs, retained along with the components, form a valuable resource for troubleshooting problems and fixing bugs, among other challenges. Many SCM systems have off-site security vaults where a copy of the latest version of the items in the controlled library is stored. These off-site vaults will prevent data loss even in the case of a disaster at the project site. Employee turnover can seriously jeopardize project schedules, since, in most cases, employees leave without transferring knowledge to their successors.

Because the SCM system maintains records of all the project-related activities performed by each team member, new employees can be brought up to speed with the help of the SCM logs and reports. Thus, implementing an SCM system increases the level of security against potential losses, unauthorized changes, employee turnover, and other disasters.

Higher Software Reuse

SCM system maintains a record of the past (project history or change history), the present (project tracking and change logs), and the future (information about planned versions and variants). The components or artifacts of the project are classified, categorized, named, and stored so that they can be easily identified and retrieved. These records help in a higher degree of software reuse, both for future versions and other projects.

Lower Maintenance Costs

When dealing with software costs, most people address only the short-term or visible costs like the costs associated with design, development, and testing. However, long-term costs associated with system operation and maintenance often constitute a large percentage (as high as 75%) of the total life-cycle cost for a given system. Blanchard [8] called this the iceberg effect, in which the initial costs are the visible part of the iceberg—the tip of the iceberg—and the operational and maintenance costs (amounting to more than 75 % of the total life-cycle costs) are the submerged part of the iceberg. So maintenance costs constitute a significant amount of a software system's total life-cycle costs. Software maintenance is usually classified as follows:

- *Corrective maintenance:* Corrects the mistakes that escaped the testing phase and are found during actual usage.
- *Adaptive maintenance:* Changes the software to perform in a new environment or with some new interfaces.
- *Perfective maintenance:* Modifies the software to include new functionality or additional features.

According to Lientz and Swanson [9], the perfective maintenance costs account for 65% of costs; 18% are adaptive; and 17% are corrective. There are many reasons for the high maintenance costs irrespective of which class they fall into. The most important among these is the absence of a proper method for handling these maintenance issues. Almost all maintenance issues involve changing something, making modifications to the existing code, or adding new code to the existing system. The people who are supposed to do these activities should have a good understanding of what they have to do, how they have to do it, where they have to make the change or modifications, and what the impact of the change will be on other programs.

If the software was developed in a systematic manner, if the documentation is perfect, the changes made to the programs are recorded, and the program dependencies are defined, then the task of the maintenance team is easy. So from the

design stage onward, proper mechanisms, ensuring that design and development are done in a systematic and controlled fashion, need to be in place. These control mechanisms will play a vital role in reducing maintenance costs.

Better Quality Assurance (QA)

One of the main objectives of a QA system is to prevent defects from occurring. In the old days when the concept of quality control (QC) was prevalent, the idea was to find defects once they had occurred. If we take the manufacturing industry as an example, the QC team was interested in finding defects before the parts were shipped. So the QC team concentrated on the final inspection with the objective that not even a single defective part got past the final inspection stage.

With the advent of the QA philosophy, the focus changed from the final inspection to the assembly line. The idea was to prevent defects from occurring rather than reject defective parts during final inspection. QA teams thus worked to identify the causes of the defects, where they were occurring, when they were occurring, and how they were occurring.

This is true in the case of software development also. In the early days, the focus was on finding errors or bugs and fixing them before the product or system was delivered to the customer. Nobody really cared about the causes of these defects and how they originated and so on. As long as they were found and fixed, life was good.

The QC philosophy had a problem, however; it was costly. A lot of time and effort could be saved if the defects were detected early. More important, though, was that in the QC approach, because nobody was looking into the causes of the defects, they remained undetected and reappeared in the next project. For example, suppose a defect was occurring because the programmer was not good at the CASE tool that he or she was using to generate the code. So every time he or she generated new code, the same mistakes were repeated. A causal analysis would have revealed this problem, and the programmer would have been trained on the tool to avoid repeating the problem. However, to do a causal analysis, one needs to have data. Accordingly, there has to be a formal mechanism for problem reporting or defect logging and tracking.

Reduction of Defects and Bugs

Once you have a defect logging and tracking system in place, once the QA teams start looking into the causes of the problems and correcting them, once the checks and audits are made to ensure that the project standards and guidelines are followed, then the number of bugs and problems will be reduced. In most cases, problems occur because the documentation is not in sync with the development, the RDD and system design document (SDD) are not updated to reflect the latest changes, and different people are using different versions of the same program or function. In many projects, there are no formal mechanisms to find out which code belongs where, what changes were made, and why and when. If the development process has a system that takes care of these types of issues, then the software development and maintenance processes will be easier.

Faster Problem Identification and Bug Fixes

In the usual system of testing, bugs are found and fixed. However, if there is a mechanism for logging bug and problem reports, categorizing them, analyzing the causes, and recording how the problem was solved, then much time and effort can be saved the next time a similar bug or problem occurs.

Also, by recording the bugs, their causes, and the corrective actions, a knowledge base will be created that will grow with time and become an invaluable resource for future tasks. When a problem is reported, the knowledge base can be searched for similar problems, and if one exists, the solution for the previous bug will help resolve the current problem faster. As mentioned previously, the knowledge base will grow in size and value as time goes on and as new problems and solutions are added to it. This also means that even when people who are working on a project leave, they leave behind the knowledge they have gained for others to use.

Process-Dependent Development Rather Than Person-Dependent Development

In the early days, when software projects were simple and small, a single individual often handled the design and details of a project. Even though projects have become larger and more complex, dependency on the individual still exists. For example, in many projects, if you remove a few key people, projects will come to a standstill, basically because the other members of the team do not have the whole picture of the project. There is no way they can have the whole picture because no documentation exists, and, even if it does exist, it is often understood only by the people who wrote it. In many cases, these documents have not been updated and are not in sync with the system that is being developed.

This kind of dependency on people is very dangerous. What happens if a key person leaves the company or is not able to work anymore? In such cases, the entire process of design and development has to start all over again, because nobody knows what to do with the current system. It is for this reason that the software engineering pioneers have always said that software development has to be process-dependent, not people-dependent.

Boehm [10] has said that talented people are the most important element in any software organization and that it is crucial to get the best people available. According to him, the better and more experienced they are, the better the chance of producing first-class results. The problem with these geniuses, however, is that their capability to work as a team member, in most cases, will not be in the same class as their talent.

Software development has become too complex and software systems so huge that it is not possible for one individual to complete a project regardless of talent. To develop software systems success fully, even the best and most talented professionals need a structured and disciplined environment, conducive for teamwork and cooperative development. According to Humphrey [11], “Software organizations that do not establish these disciplines condemn their people to endless hours of repetitively solving technically trivial problems. There may be challenging work to do, but their time is consumed by mountains of uncontrolled detail. Unless these details

are rigorously managed, the best people cannot be productive. First-class people are essential, but they need the support of an orderly process to do first-class work."

Assurance That the Correct System Has Been Built

Software development, as we have seen, starts with the requirements analysis. We have also seen that during the software development life cycle, the requirements will undergo many changes. So how do we make sure that the system that is being delivered to the customer is what the customer initially wanted and contains all the changes that were suggested during the development period? In other words, how do clients or customers know that what they are getting is what they asked for?

There should be some sort of process for documenting the initial requirements and the changes made to them. There should also be a mechanism for checking or auditing the software system or product that is being delivered to the customer and certifying that the product satisfies the requirements. In other words, there should be a facility to conduct audits (or reviews) to ensure that what is developed and delivered is complete in all respects and exactly what was specified.

Summary

Software systems are becoming more and more complex and sophisticated and are being used increasingly in mission-critical applications. Due to the changing nature of software and software development, it is necessary to have a system in place to manage and control change; otherwise, chaos and confusion may ensue, resulting in low quality, lower productivity, and even the scrapping of a project. One weapon against such an outcome is QA, an important element that can help to reduce bugs and maintenance costs. To be successful in the long run, however, we need a process-dependent system.

SCM is an ideal solution for the issues discussed and an excellent foundation from which other process-improvement methodologies can be launched. SCM provides a mechanism for managing, documenting, controlling, and auditing change. Accordingly, in today's complex software development environment, SCM is a must for all projects, irrespective of their nature, size, and complexity. Furthermore, it is better to have the SCM functions in place as early as possible. According to Davis [12], SCM procedures should be designed and approved and recorded in a document—the SCM plan. This document should be written early in a project (as described in Chapter 2—as early as the project start-up phase), typically getting approved around the same time that the software requirements specifications are approved.

References

[1] Musa, J. D., "Software Engineering: The Future of a Profession," *IEEE Software*, Vol. 22, No. 1, 1985, pp. 55–62.

[2] Boehm, B. W., and P. N. Papaccio, "Understanding and Controlling Software Costs," *IEEE Trans. Software Engineering*, Vol. 14, No. 10, 1988, pp. 1462–1477.

[3] Jones, C. T., *Estimating Software Costs*, New York: McGraw-Hill, 1998.
[4] Jones, G. W., *Software Engineering*, New York: John Wiley and Sons, 1990.
[5] Lehman, M. M., and L. Belady, *Program Evolution: Processes of Software Change*, London: Academic Press, 1985.
[6] Lehman, M. M., and L. Belady, "A Model of Large Program Development," *IBM Syst. J.*, Vol. 15. No. 3, 1976, pp. 225–252.
[7] Lehman, M. M., "Software Engineering, the Software Process and Their Support," *Software Engineering J.*, Vol. 6, No. 5, 1991, pp. 243–258.
[8] Blanchard, B. S., *System Engineering Management*, New York: John Wiley & Sons, 1991.
[9] Lientz, B. P., and E. B. Swanson, *Software Maintenance Management*, Reading, MA: Addison-Wesley, 1980.
[10] Bohem, B. W., *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall, 1981.
[11] Humphrey, W. S., *Managing the Software Process*, New York: Addison-Wesley, 1989.
[12] Davis, A. M., *201 Principles of Software Development*, New York: McGraw-Hill, 1995.

CHAPTER 5

SCM: Basic Concepts

Introduction

SCM is the set of activities that are performed throughout the project life cycle—from requirements analysis to maintenance. SCM is important because software is subject to constant change—software systems undergo changes when designed, when built, and even after being built. Uncontrolled and unmanaged change can create confusion and lead to communications breakdown problems, shared data problems, multiple maintenance problems, simultaneous update problems, among others. So change has to be controlled and managed.

A software development project produces the following items:

- Programs (e.g., source code, object code, executable programs, component libraries, functions, and subroutines);
- Documentation (e.g., requirements definition, systems analysis, systems design, high-level design, low-level design, test specifications, test plans, test scripts, installation manuals, release notes, and user manuals);
- Data (test data and project data).

These items are collectively called a software configuration. IEEE [1] defines a software configuration as the functional and physical characteristics of the software as set forth in technical documentation or achieved in a product.

The SCM system[1] identifies these items (the software items) and records their properties and relationships. This task would be very easy if the items and the systems were not subject to change. Unfortunately, however, that is not the case.

Changes can occur at any time. Bersoff's [2] first law of system engineering states that no matter where you are in the system life cycle, the system will change and the desire to change it will persist throughout the life cycle. So we have to deal with change and SCM does that. That is not the only thing that SCM does, however; it also—through audits and reviews—ensures that the items that are being released satisfy the requirements that were set forth in the requirements and design documents.

1. SCM system refers to the tools, plans, and procedures that collectively implement SCM in the project. It is the collection of all activities and personnel and other resources that performs SCM.

Accordingly, we can define SCM as the set of activities whose main purposes are to identify the CIs (the items that are supposed to change or that will undergo change); to find the properties, characteristics, and interdependencies of these items and record them; to monitor these items; to manage the changes made to these items; to document and report the change process; and to ensure that the items delivered are complete and satisfy all requirements.

Overview of SCM

To identify, control, and manage change, one must first identify which items in the project will be subject to change. Therefore, we must first identify the items that we plan to control and manage. These items are called, in SCM terminology, CIs. IEEE [1] defines CIs as an aggregation of hardware, software, or both that is designated for CM and treated as a single entity in the CM process. Examples of such entities include a program, a group of programs, a component library, a function, a subroutine, project documentation, a user manual, a test plan, test data, and project data. The SCM system is supposed to record the functional and physical properties (e.g., the features, what it is supposed to do, what performance criteria it is supposed to achieve, the size, and lines of code) of these CIs. Some examples of CIs in a project are listed as follows:

- The project plan;
- The SCM plan;
- The RDD;
- The analysis, design, coding, testing, and auditing standards;
- The SAD;
- The system design document (SDD);
- The prototypes;
- The HLD document;
- The LLD document;
- The system test specifications;
- The system test plan;
- The program source code;
- The object code and executables;
- The unit test specifications;
- The unit test plans;
- The database design documents;
- The test data;
- The project data;
- The user manuals.

This list is by no means exhaustive. It varies from project to project. The designers of the SCM system for a particular project decide which items should be CIs. The characteristics of each CI and their interdependencies with one another are recorded. Usually, this information is recorded in what is called a CM database or

in repositories in the case of SCM tools. We will learn more about the CM database later in this chapter.

Once the CIs are identified and their characteristics recorded, the next steps in SCM, as shown in Chapter 1, are configuration control, status accounting, and configuration audits. We look briefly at these activities later in this chapter. Before we proceed, however, we should familiarize ourselves with some of SCM terminology and concepts that we will be using in this book, including baselines, deltas, versions, variants, branches, builds, and releases.

Baselines

Baselines play a very important role in managing change. During the software development process, the CIs are developed. For example, design documents are created, programs are coded and tested, and user manuals are prepared. When a CI is complete, it is handed over to the CM team for safekeeping. The CM team will check whether the item that is given to them is complete (or contains all the necessary components) per the SCM plan and place it in the controlled library. During specific points during the software development, usually after the requirement phase, design phase, and release phase, the set of CIs under SCM control are formally designated as a baseline. So a baseline need not contain all the items that are in the controlled library. Which items are to be included in a baseline is depends on the purpose for which the baseline is created. For example, a release baseline will only contain the CIs that are to be given to the customer.

There are two schools of thoughts regarding baselines. One suggests that the required CIs are placed into one of the different baselines. The other proposes that a CI becomes its own baseline once it is given over to SCM for control. The problem with declaring any configured item as a "baseline" is that one loses the distinction between something that is merely checked in, and when it becomes a part of some deliverable (such as the development, test, or deployed or release baselines). Just checking an artifact in may not even mean it gets included in a build. It may be an interim step as part of a parallel development effort, for example. A baseline does not represent simply any changed item; rather, it is a particular grouping of items at a specific point in time, for a given purpose.

Baseline is an SCM concept that helps to control change. IEEE [1] defines baseline as a specification or product that has been formally reviewed and agreed on, which, thereafter, serves as the basis for further development and can be changed only through formal change control procedures. So once a baseline is established, the CIs in the baseline can be changed only through a formal change management process. Pressman [3] compares the process of change management to a room with two doors: the IN and OUT doors. According to him, when an item has passed through the IN door—to the controlled environment—the item is brought under configuration control. Then the only way to make changes to the item is get it out through the OUT door—using formal change management methods.

The baseline is the foundation for CM. The definition of SCM contains the concept of identifying the configuration—the functional and physical characteristics—of each CI at discrete points in time during the life cycle process. *The configuration*

of software at a discrete point in time is known as a baseline. Each baseline serves as a point of departure or reference for the next development stage.

A software baseline is a set of software items formally designated and fixed at a specific time during the software life cycle. A baseline, together with all approved changes to the baseline, represents the current approved configuration. Usually, baselines are established after each life cycle phase at the completion of the formal review that ends the phase. Thus we have such baselines as the requirements baseline, the design baseline, and the product baseline, as shown in Chapter 2.

A baseline provides the official standard or point of reference on which subsequent work is based and to which only authorized changes are made. After an initial baseline is established, every subsequent change made to the CIs is done using the configuration control process or, in other words, using formal change management procedures. Whenever an item is changed, all the processes involved in making the change—change initiation to change requests[2] to change disposition and implementation—are recorded. Then, the item that is being changed is reviewed and saved as a new version of the item. For all these change management processes, the baseline serves as a reference point.

Baselines should be established at an early point in the project. However, bringing all items under configuration control too early will impose unnecessary procedures and will slow the programmers' work. This is because before a software CI becomes part of a baseline, changes may be made to it quickly and informally. For example, consider a programmer developing a program. After he or she has completed coding, while doing the unit testing, the programmer stumbles on a better algorithm to accomplish some task in the program. Because the program is not part of a baseline, the programmer can make the necessary change to the program, recompile it, and continue with the testing. However, if the same situation occurs after the program is checked-in or has become part of a baseline, then the programmer will have to make a change request and follow the change management procedures to make the change.

So when should baselines be established? There are no hard and fast rules on this issue. It depends on the nature of the project and the thinking of the SCM system designers (the people who designed the SCM system and wrote the SCM plan). Establishing baselines involves a trade-off between imposing unnecessary procedures (thus reducing productivity) and letting things go uncontrolled (which will result in project failure). So these two factors should be kept in mind when deciding when to baseline. As long as the programmers can work on individual modules with little interaction, a code baseline is not needed. As soon as integration begins, formal control is essential.

So prior to a CI becoming a controlled item, only informal change control[3] is applied. The developer of the item can make whatever changes are justified by project

2. Change request (CR) is a request to make a change or modification. A CR form is a paper or electronic form that is used to initiate a change and that contains the details of the change such as the name of the change originator, the item to be changed, and the details of changes.
3. Informal change control is applicable when the developers can make changes to their programs without following the SCM procedures. This is possible when the item has not been checked in and is not under SCM control.

and technical requirements, as long as these changes do not conflict with the system requirements.[4] However, once the object has undergone formal technical review and has been approved, a baseline is created. Once the CI becomes a baseline, project level change control or formal change control is implemented.

Even though the most common baselines are the requirements, design, and product baselines, a baseline can be established whenever a need is felt. For example, many projects have many more baselines than the three mentioned above. A new baseline is not necessarily established each time a CI is modified or added to the library. The number of baselines that a project has is determined by the SCM team, the project's needs, and what is advantageous. The information about when to create a baseline will be in the SCM plan.

You can have a baseline at the start of each phase, if you so choose. As for trade-offs, the only time there is a real need to establish a baseline is, if after it is done, a copy of the baseline leaves SCM control in some fashion (i.e., to a test lab or deployment)—that is the minimum for creating a baseline. You must know the "set" of software that starts a phase or is deployed so that you know what changes should be made to it later. In considering how to control the number of baselines, ask the following questions:

- What are the requirements from the customer vis-à-vis providing demonstrable progress reporting?
- What are the needs and requirements internal to your organization or project about status and progress reporting?
- What metrics do your customers, management, and project leadership need to see?

The answers to these questions can help you establish a baseline schedule.

Check-In and Check-Out

We have seen that once a CI is developed, reviewed, and approved, it is kept in a controlled library or repository. This process of reviewing, approving, and moving an item into the controlled environment is called check-in.

Once an item is checked in, it becomes a controlled item, and all change management procedures apply to it. It cannot be taken out and modified whenever a programmer feels like doing so, even if he or she is the author or developer of the item.

For making changes to an item that is in the controlled library, the change management process, which is discussed in detail in Chapter 7, must be followed. That is, a change request has to be submitted and approved, among other things. Once the change request is approved, the configuration manager will copy the item (and

4. Developers can find out whether the changes they make to their programs conflict with the system requirements by going through the system design specifications and HLD documents. For example, in a program, whether the developer first calculates the tax and subtracts it from the earnings or multiplies the earnings with the (1-tax/100) does not conflict with the system requirements. This is a decision that the programmer takes according to his or idea about which is the best algorithm.

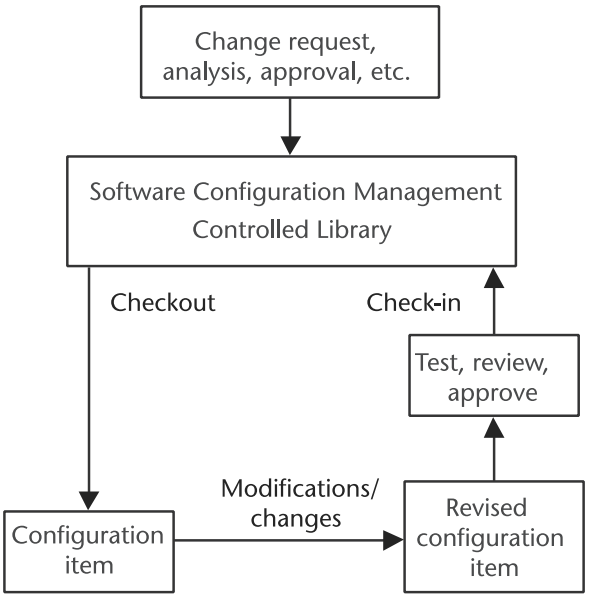


Figure 5.1 Check-in and check-out.

other impacted items, if any) from the controlled library so that modifications can be made. This process is called check-out. Thus, to make a change to an item that is under SCM control, it has to be checked out of the controlled library. After the changes are made, the item or items are again tested and reviewed and, if approved, proceed to be checked in to the controlled environment. The check-out/check-in process is shown in Figure 5.1. Today's SCM tools have made this whole process of check-in and check-out an easy task. Many tools allow programmers to work on CIs without physically checking out the items. Also, today's SCM tools allow more than one person to simultaneously work on the same CI (concurrent development). These facilities provided by modern SCM tools are discussed later in this chapter.

Versions and Variants

During the software development life cycle, the CIs evolve until they reach a state where they meet the specifications. This is when the items are reviewed, approved, and moved into the controlled environment. However, we have seen that the story does not end there. The item will undergo further changes (due to various reasons such as defects and enhancements) and to make those changes, the change control procedures must be followed. The items have to be checked out, and the changes need to be implemented, tested, reviewed, approved, and again moved back to the controlled library. This change process produces a new version or revision of the item.

A version is an initial release or rerelease of a CI. It is an instance of the system that differs in some way from the other instances. New versions of the system may have additional functionality or different functionality. Their performance characteristics may be different, or they may be the result of fixing a bug that was found by the developer, tester, user, or customer.

Some versions can be functionally equivalent but may be designed for different hardware or software environments. In such cases, they are called variants. For example, two different instances of the same item—e.g., one for Windows and the other for Linux—can be called variants rather than different versions. Unlike a version, one variant of an item is in no sense an improvement on another variant.

As we have seen, the items once moved into the controlled environment can be changed only by using SCM change control methods. Each such change produces a revision or version. So each change to a controlled item produces a new version, and, except for the first, each version has a predecessor. Similarly, except for the most recent, each version has a successor. The different versions of an item represent its history. It explains how an item got transformed or evolved from its initial form or stage to its current form. Usually, a new version of an item is created by checking out the most recent copy and making changes to it.

Parallel Development and Branching

So far we have seen that an item is checked out, and changes are made and then tested, reviewed, approved, and checked in. So the versions will form a linear line as shown in Figure 5.2.

In real life, however, this linear development might not always be possible. In such cases we use what is called a branch. Branches (Figure 5.3) are deviations from the main development line for an item. They are a convenient mechanism for allowing two or more people to work on the same item at the same time—parallel, concurrent development—perhaps for different goals. A common scenario is having one person working to add new features to the product, while a second is doing bug fixes on prior versions.

The version numbers of branches can be a little confusing, so they warrant a quick discussion. Version numbers on the main development line have only two parts—a major and minor number. Branches have four parts to their numbering scheme. The first two parts represent the point at which the branch splits off the main line. The third number indicates which of the many possible branches it is. For example, in Figure 5.3, we have only one branch originating from 1.3. As such,

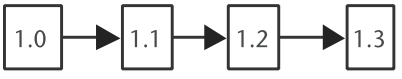


Figure 5.2 Version numbers.

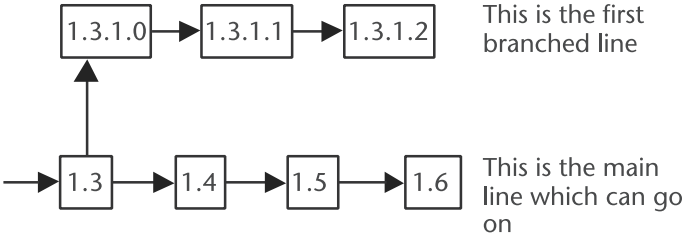


Figure 5.3 Branching for parallel development.

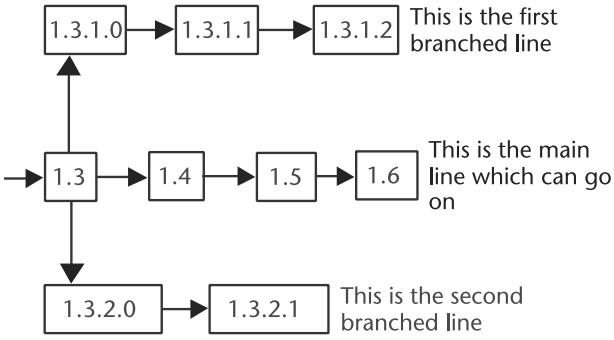


Figure 5.4 Multiple branches.

its numbering starts at 1.3.1.0 and proceeds from there. If a second branch is later formed from 1.3, then its numbering will begin with 1.3.2.0, as shown in Figure 5.4.

Branches can also extend from existing branches. For example, a branch can be formed from 1.3.1.1. This branch will have a six-part numbering scheme starting with 1.3.1.1.1.0 as shown in Figure 5.5. The first four parts represent the point at which the branch split off from the parent branch. The fifth number indicates which of the many possible branches it is. For example, in Figure 5.5 we have only one branch originating from 1.3.1.1. As such, its numbering starts at 1.3.1.1.1.0 and proceeds from there. If a second branch is later formed from 1.3.1.1, then its numbering will begin with 1.3.1.1.2.0.

Branches are often used as a temporary means of allowing parallel and concurrent development on a single file. Sooner or later, the edits made to the branched line must be incorporated into the main evolutionary line for the file. When doing this, the changes made by the different persons have to be merged. If the changes are made at different parts of the item, then the merge is an easy task. However, if two people have changed the same lines of an item, then a decision has to be reached about how the merge is to take place. The person who does the merging should decide which one to keep and which one to discard. SCM tools have automatic merging facilities that allow interactive merging, in which the tool will compare the changed portions of the two changed files to the original file (usually called the

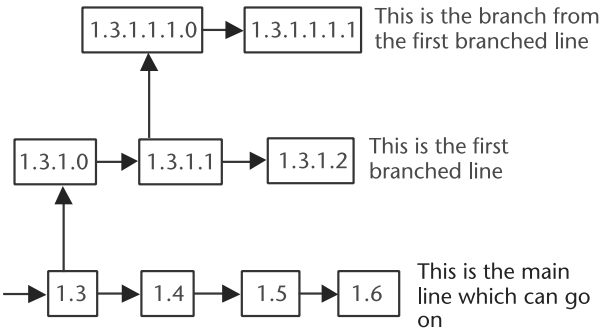


Figure 5.5 Branch from an existing branch.

ancestor), and the user can choose which change to accept. In the author's opinion, however, it is always better to do an interactive or manual merge, because human judgment is better that the judgment afforded by the algorithms used by the tools.

After the merge occurs, the branches have outlived their utility and no longer need to evolve separately.

Naming of Versions

We have seen that all CIs have to be named and that the name has two parts: a number part, which changes with each version, and the name of the item (including the project, type, and other details). The names should be descriptive. For example, in a large project with many modules and subsystems, a simple name for a CI is not a good idea. In such cases, the name should have elements of the project, subsystem, module, and other components in it so that it can be easily identified with a project or subsystem. The number part of the name should be designed in such a way as to determine its relative position in the version hierarchy. For example, we have seen that the first version will be identified as 1.0 and subsequent revisions 1.1, 1.2, 2.0, 2.1, and so on. A branch from version 1.1 will be identified as 1.1.1.0 and its subsequent versions 1.1.1.1 and so on. A second branch will be identified as 1.1.2.0 and so on.

Source and Derived Items

An item that is created from another item or set of items is called a derived item. The items from which a derived item is created are called source items. For example, an executable program is a derived item—an item that is derived from the source code using a compiler. In the case of derived items, the details, such as the list of source items used to derive the item, the tool or tools used to derive the item, and the environment in which the derivation was done, are important and should be documented. This is important from the point of view of reproducibility and repeatability. For example, if you want to derive the particular version of a derived item, you will need all the above information to do so. Also, if a problem is detected with a version of the derived item, then the following questions need to be answered to solve the problem:

- Which versions of which source elements (name and version of each source item) were used to build the item?
- Which tools were used for the process?
- What environment variables and parameters (such as compile and link-edit options) were used to by the tools while building the item?

These are the first questions one must answer if a problem occurs, and a good SCM system should be able to provide the answers to these questions, as these are the answers that fully describe a derived item.

System Building

System building is the process of combining "source" components of a system into components, which execute on a particular target configuration. The system or parts of it have to be rebuilt after every change in the "source." The following factors must be considered:

- Have all components that make up the system been included in the build instructions (e.g., dependencies resolved and include paths set), and do they have the proper version?
- Are all required ancillary files (data and documentation, e.g.) available on the target machine?
- Are the required tools (e.g., the compiler or linker) available, and do they have the right version?

The system is built using a command file that specifies such factors as the components of the system (both source and derived), their versions, their location in the controlled environment, the system building tools (e.g., a compiler and a linker) and their versions, and the options and environmental parameters that were set. In the IBM mainframe, this file is usually a JCL file; in UNIX, it is a shell script; in modern integrated development environments such as Visual C++ and Visual Basic, it is a make file or project file. These command files are also CIs and are necessary for reproduction of the particular configuration.

The build management facility of many SCM tools automates the process of constructing the software system and ensures that the systems are built completely and accurately at any time. These configuration builders save time, shorten build cycles, and eliminate build errors by providing repeatable, automated builds of the software development projects. In most cases, the system building tools work with the version management tools and extract the correct versions of development objects from libraries. This simplifies the system building process and eliminates errors when building complex versions on multiple operating systems.

Releases

A release consists of more than just the executable code. It includes installation files, data files, setup programs, and electronic and paper documentation. A system release is the set of items that is given to the customers. Each system release includes new functionality or features or some fixes for the faults found by customers, developers, or testers. Usually, there are more revisions of a system than system releases. Revisions or versions, as mentioned before, are created for internal use and may never be released to customers. For example, a revision may be created for testing.

When a release is produced (using the system building process), as we have seen, it is important to record the environment in which it was produced, including the operating system, versions of the components used, and other parameters such

as compile and link-edit options. This is important from the SCM point of view, because at a later stage, it might become necessary to reproduce the exact configuration that was released. For example, consider a bug that is discovered after the system is released. The easiest way to find the source of the problem is to find out what components were changed. The problem could be either due to a bug in one of the changed components or due to some environmental variables being changed (like some compiler or linker options). Accordingly, if we have a record of the components and the environment details used for the release, then it is easy to track the source of the defect. One merely needs to compare the details of the current release with the previous release and see what has been changed. Thus, it is imperative that a proper mechanism to record the details of each and every release be instituted.

A release to a client or a system release should contain identifiers indicating the release or version number. It should also include a release note containing the following information:

- Installation requirements (e.g., required operating system, memory, and processor specifications);
- How to install the system and how to test the system to ensure that the installation was successful;
- How to upgrade from an earlier version of the system;
- The key or serial number of the product, if such a number is required for installation;
- A list of known faults and limitations of the particular version of the system and a list of the faults that were fixed in the current release;
- New features introduced in the release;
- Instructions for contacting the supplier of the system for technical support or if problems arise.

Today, most of the above-mentioned activities, including the registration of the product, are done by installation programs, so release notes are not as important as they once were.

Deltas

In an ideal situation, all changes made to the CIs should be recorded and all of the different versions of the items should be kept, because in a software system not all users will be using the latest version. So, even though the system may be in version 6.0, some users will still be using version 1.0 or 2.1. Accordingly, CM systems should be able to produce the details of the latest as well as past versions and to reproduce the components of every version. For example, years after a system is released, a request for a component in the first version can come up. Even if the system is currently in its seventh or eighth version, companies cannot ignore a client that still uses the initial version.

Ideally, copies of all versions should be in a repository. However, this is not practical, because of the amount of disk space required. Instead, we create what is

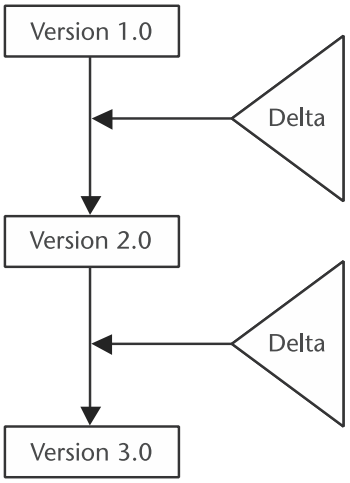


Figure 5.6 Use of deltas.

known as a delta. When a new version is created, the difference between the new and the previous version is called delta. With this method, instead of storing full copies of all versions, one version and the deltas are stored, so that at any point in time the required version can be derived by applying the relevant deltas to the base version. Figure 5.6 illustrates the concept of deltas.

Deltas are smaller than the source code of a system version, so the amount of disk space required for version management is greatly reduced. The two types of delta storage are forward deltas and reverse deltas, as shown in Figure 5.7.

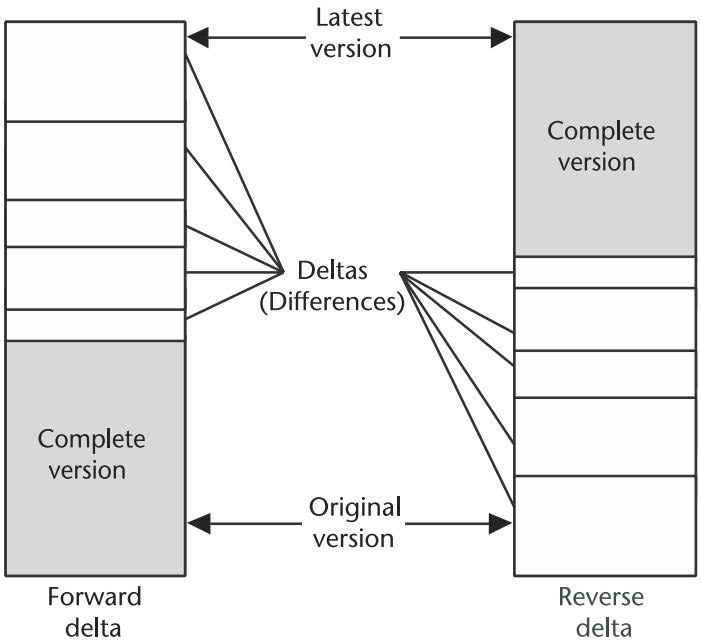


Figure 5.7 Forward and reverse deltas.

The principle of forward delta storage is that the system maintains a complete copy of the original file. After this, whenever a new version is checked in, the two versions are compared and a delta report is produced. Then, this delta report is stored instead of storing the full copy of the new version. Whenever the new version is required, the delta is applied to the original to get the new version.

In the case of reverse delta storage, only the most recent version of the module is kept in the complete form. Whenever a new version is checked in, it is compared to the previous version and the delta is created. Then, the previous version is deleted, and the new version is stored.

The problem with forward delta storage is that as more revisions are added, more computation is required to obtain the latest revision. So the greater the number of revisions, the longer the retrieval time will be. This is because in the case of the forward delta, the change manager must always start with the original version and then apply the deltas one at a time to create the latest version. In the case of the reverse delta option, no computation is required to get the latest version, because it is stored in its full form.

Using deltas is a classical space-time trade-off—deltas reduce the space consumed but increase access time. However, an SCM tool should impose as little delay as possible on programmers. Excessive delays discourage the use of version controls, or induce programmers to take shortcuts that compromise system integrity.

The decision to go in for delta storage as opposed to storing copies of all versions should be made with respect of amount of storage space required and cost of storage and retrieval. With the cost of storage systems coming down, storing copies of all versions can be considered, thus eliminating the need of deltas. Another factor that decides whether or not to use delta storage is the accuracy in rebuilding the required versions using the deltas vis-à-vis the cost of storage and the time taken to retrieve an archived item from the backup mechanism. Whether to use delta storage or not is a subject of debate among SCM professionals. In my opinion, if the complete copies can be stored at a reasonable price and be retrieved as fast as recreating using the delta storage, then it is better to eliminate delta storage or to limit it to text files.

The decision to go in for forward delta storage or reverse delta storage depends on the nature of the project. If the latest versions are more frequently required, then it is always better to use the reverse delta. In the real world, more than 75% of archive accesses are for the latest version, and this explains the popularity of reverse delta storage among the change management tools. Many tools also offer the facility to use delta storage or keep the complete copies. So depending on the nature, size, and type of CIs in the project, tool administrators can select a method to suit their preference.

SCM Database

We have seen that the properties, characteristics, and interdependencies of the CIs should be recorded in a database—the CM database. The CM database is used to record all relevant information related to configurations. The principal functions of such a database are to assist in assessing the impact of system changes and to provide information about the SCM process. The CM database, in addition to the

details about the CIs, contains information about change requests (which are also CIs), their status, and information regarding the review and audit processes.

The contents and structure of the CM database should be defined during the SCM system design stage and should be documented in the SCM plan. In modern CASE environments, the configuration database is part of the system and the details of the items are automatically recorded. In the case of manual SCM systems, the details have to be entered manually into the system. The system should have necessary precautions and safety mechanisms to prevent bypassing the procedures and thus corrupting the data and making it useless. If a CI is added without an entry in the database, then the integrity of the data in the database and the usefulness of the database are lost.

A CM database should be able to provide answers to queries such as the following:

- What is the current configuration? What is its status?
- Which person has taken delivery of a particular version of the system?
- What hardware and operating system configuration are required to run a given system version?
- How many versions of a system have been created and what were their creation dates?
- What changes have been made to the software, documentation, and other items in the project? Who made them, and when were they made?
- Were the changes approved by somebody or just informally done?
- What versions of a system might be affected if a particular component is changed?
- How many change requests are pending on a particular item?
- How many reported faults exist in a particular version?
- Can I recreate the original from the changed version or the changed version from the original?
- Can I find out what happened to a specific item at some point in time, such as what changes were made to it and so on?
- Does my change affect others?

With the increasing popularity of SCM tools, the necessity for a configuration database is decreasing. SCM tools have their own repositories where they can store SCM-related information. The advantage of these systems is that SCM information is captured automatically as and when each activity is performed. So there is no need to enter details manually in such situations as when a change request is initiated, when an item is released, or when a change is made. This feature saves considerable time and effort and reduces the chance of creating errors that can occur during manual data entry. Also, with the facility to automatically capture SCM information, such as when the activities happen, the SCM tool user has the ability to capture comprehensive SCM information without any additional effort.

SCM Activities

Configuration identification, the first activity of CM, is the process of defining each baseline to be established during the software life cycle and describing the software

CIs and their documentation that make up each baseline. First, the software must be grouped into CIs. Once the CIs and their components have been selected, some way of designating the items must be developed. This is done by the development of a numbering and naming scheme that correlates the code and data items with their associated documentation. Finally, each CI must be described by the documentation in terms of its functional, performance, and physical characteristics.

Configuration control is the process of evaluating, coordinating, and deciding on the disposition of proposed changes to the CIs and includes implementing approved changes to baselined software and associated documentation. The change control process ensures that changes that have been initiated are classified and evaluated, approved, or disapproved, and that those approved are implemented, documented, and verified.

Configuration status accounting is the process used to trace changes to the software. It ensures that status is recorded, monitored, and reported on both pending and completed actions affecting software baselines.

Configuration auditing is the process of verifying that a deliverable software baseline contains all of the items required for that delivery and that these items have themselves been verified to determine that they satisfy requirements.

We will examine these activities in greater detail in Chapters 6–9.

Summary

This chapter details the various CM concepts and SCM terminology. It explains SCM activities and how they relate to one another. Further, it provides readers with an understanding of the fundamental concepts of SCM such as versions, variants, branching, merging, deltas, system building, and releases.

In today's business environment, where competition is fierce and one has to constantly improve and innovate to stay ahead of the competition, SCM can give an organization a strategic advantage over its competitors. An organization that uses SCM has a definitive edge over others that do not use SCM or that use it inefficiently or ineffectively.

References

[1] IEEE, *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610–1990), IEEE Standards Collection (Software Engineering)*, Piscataway, NJ: IEEE, 1997.

[2] Bersoff, E. H., V. D. Henderson, and S. G. Siegel, *Software Configuration Management: An Investment in Product Integrity*, Englewood Cliffs, NJ: Prentice-Hall, 1980.

[3] Pressman, R. S., *Software Engineering: A Practitioner's Approach (5th Edition)*, New York: McGraw-Hill, 2001.

Selected Bibliography

Babich, W. A., *Software Configuration Management: Coordination for Team Productivity*, Boston, MA: Addison Wesley, 1986.

Ben-Menachem, M., *Software Configuration Guidebook*, London: McGraw-Hill International, 1994.

Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, 1992.

CM Crossroads™: The Configuration Management Community (http://www.cmcrossroads.com/).

Conradi, R. (ed.), *Software Configuration Management: ICSE'97 SCM-7 Workshop Proc.*, Berlin: Springer-Verlag, 1997.

IEEE Standards Collection, *Software Engineering*, New York: IEEE, 1997.

Magnusson, B. (ed.), *System Configuration Management: ECOOP'98 SCM-8 Symp. Proc.*, Berlin: Springer-Verlag, 1998.

Pressman, R. S., *Software Engineering: A Practitioner's Approach*, New York: McGraw-Hill, 1997.

Sommerville, I., *Software Engineering*, Reading, MA: Addison-Wesley, 1996.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

CHAPTER 6

Configuration Identification

Introduction

Configuration identification is the basis for subsequent control of the software configuration. The software configuration identification activity identifies items to be controlled, establishes identification schemes for the items and their versions, and establishes the tools and techniques to be used in acquiring and managing controlled items. These activities provide the basis for the other SCM activities [1].

The configuration identification process involves the selection, designation, and description of the software CIs. Selection involves the grouping of software into CIs that are subject to CM. Designation is the development of a numbering or naming scheme that correlates the software components and their associated documentation. Description is the documentation of functional, performance, and physical characteristics for each of the software components.

IEEE [2] defines configuration identification as an element of CM, consisting of selecting the CIs for a system and recording their functional and physical characteristics in technical documentation. In other words, configuration identification is the process whereby a system is separated into uniquely identifiable components for the purpose of SCM. This is the first major SCM function that has to be started in a project (the design of the SCM system and the preparation of the SCM plan are done before configuration identification begins). Effective configuration identification is a prerequisite for other SCM activities, all of which use the products of configuration identification.

The software under control is usually divided into CIs, also known as computer software CIs (CSCIs). A CI is an aggregation of software that is designated for CM and is treated as a single entity in the SCM process. In other words, CI is the term used for each of the logically related components that make up some discrete element of software. A variety of items, in addition to the code itself, are controlled by SCM. Software items with potential to become CIs include plans, specifications and design documentation, testing materials, software tools, source and executable code, code libraries, data and data dictionaries, and documentation for installation, maintenance, operations, and software use. For example, if a system contains several programs, each program and its related documentation and data might be designated a CI. Determining what characteristics must be captured so that the properties and requirements of the product are correctly reflected is an important decision.

The configuration identification process should capture all characteristics of the software to be controlled—its content, the content of documents that describe

it, the different versions as the contents are changed, the data needed for operation of the software, and any other essential elements or characteristics that make the software what it is. A CI can contain other CIs and these are sometimes referred to as computer software components (CSCs) and computer software units (CSUs). A CSC is a functional or logically distinct part of a computer software CI. CSCs may be top-level (TLCSCs) or lower-level (LLCSCs). A CSU is the smallest logical entity or the actual physical entity of the system.

The software product shall be organized as one or more CSCIs. Each CSCI is part of a system, subsystem, or prime item and shall consist of one or TLCSCs. Each TLCSC shall consist of LLCSCs or CSUs. LLCSCs may consist of other LLCSCS or CSUs. TLCSCs and LLCSCs are logical groupings. CSUs are the the smallest logical entities and the actual physical entities implemented in code. The static structure of CSCIs, TLCSCs, LLCSCs, and CSUs shall form a hierarchical structure as illustrated in Figure 6.1. The hierarchical structure shall uniquely identify all CSCIs, TLCSCs, LLCSCs, and CSUs.

To accomplish configuration identification, it is necessary to perform the following steps:

- Develop the criteria for selecting items to be placed under configuration control.
- Select CIs and define the relationships between the CIs.
- Establish a software item hierarchy (structure and elements of the complete software system) and define the interrelationships between the CIs. The

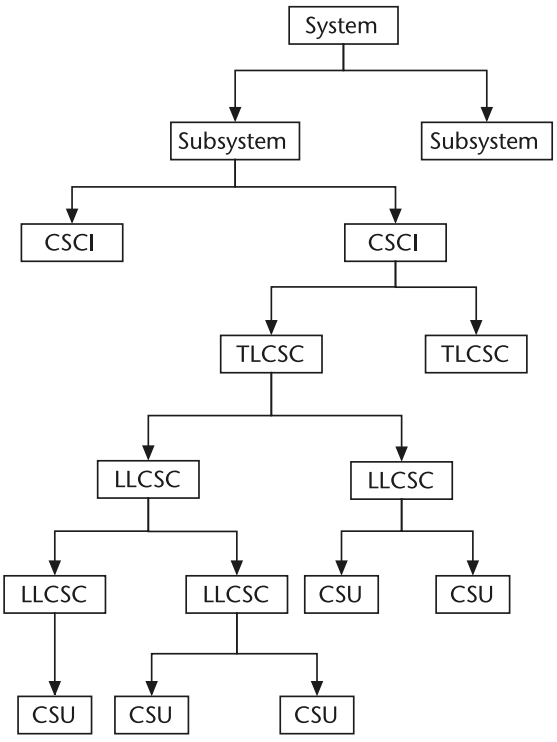


Figure 6.1 CSCI structure.

structural relationships among the selected CIs, and their constituent parts, affect other SCM activities or tasks, such as software building or analyzing the impact of proposed changes. Proper tracking of these relationships is also important for supporting traceability verifications.
- Develop an identification or naming scheme for identifying the CIs clearly and unambiguously.
- Select configuration documentation to be used to define configuration baselines for each CI.
- Establish a release/version management system for configuration documentation.
- Define and document interfaces to and between CIs.
- Define and establish baselines to be used.
- Establish the procedure for a baseline's acquisition of the items.
- Assign identifiers to CIs and their associated configuration documentation, including version numbers where appropriate.
- Check that the marking or labeling of items and documentation with their applicable identifiers enable correlation between the CI and other associated data. The documents that capture the functional and physical characteristics of the CI and the corresponding documentation will become part of the CI. The naming of those items should be consistent with that of the CI for easy association. For example, if the name of the documentation for a program PGM XYZ is DOC XYZ it will help in better correlation.
- Ensure that applicable identifiers are embedded in the software and on its storage media. For example, the source code of the program named PGM XYZ can contain the name PGM XYZ. (Consider a COBOL program named PGM XYZ; here the program name can be embedded in the source code as PROGRAM ID in the IDENTIFICATION DIVISION.)

There are no hard and fast rules as to the order in which the above steps must be performed. It depends on the organization, project, and the SCM system designers and implementers.

EIA-649 [3] identifies configuration identification as the basis from which the configuration of products are defined and verified; products and documents are labeled; changes are managed; and accountability is maintained. Configuration identification affords many benefits and purposes, including the following:

- To select products based on functionality, verifiability, supportability, complexity, risk, and management activity;
- To determine the structure (hierarchy) of a product and the organization and relationships of its configuration documentation and other product information;
- To document the performance, interface, and other attributes of a product;
- To determine the appropriate level of identification marking of product and documentation;
- To provide a unique identity to a product or to a component part of a product;
- To provide a unique identity to the technical documents describing a product;
- To modify the identification of product and documents to reflect incorporation of major changes;

- Maintenance of release control of documents for baseline management;
- To distinguish between product versions;
- To correlate a product to related user or maintenance instructions;
- To manage information including that in digital format;
- To correlate individual product units to warranties and service life obligations;
- To correlate document revision level to product version/configuration;
- To provide a reference point for defining changes and corrective actions.

Impact of CI Selection

Selecting CIs is an important process that must achieve a balance between providing adequate visibility for project control purposes and providing a manageable number of controlled items. Poor CI selection can adversely affect costs and scheduling and can become an unnecessary administrative and technical burden during and after software development. The number of CIs in a system is a design decision—a decision made by the people who design the SCM system. They are the people who decide which items should be brought under configuration control.

This process should be done carefully, because the selection of CIs needs to be correct: You should not select more or less than what is necessary. However, with the introduction of SCM tools, this issue—the number of CIs—is not very important. Modern tools can efficiently and effortlessly handle any number of CIs without any problems. Still, the tools can only reduce the workload of managing the CIs; other problems, such as reduced visibility, too much documentation, and inefficient design, cannot be solved by the tools. Hence, the selection of the CIs is still important. Technology can only help do things like automating processes, performing repetitive and monotonous tasks, and reducing the workload of managing the CIs. The question of what items should be made into CIs is still a management decision based on experience and judgment—a decision that can be made only by a person or group of persons who have the required knowledge about the project and who have the relevant experience and expertise.

Effects of Selecting Too Many CIs

Selection of too many CIs may result in hampered visibility and management rather than improved control. Examples of such difficulties include the following:

- Increased administrative burden in preparing, processing, and reporting of changes, which tends to be proportional to the number of CIs.
- Increased development time and cost as well as potential creation of an inefficient design. When there are too many CIs, the documentation and other procedures increase and take valuable time that could be devoted to design and development. As a result, too many CIs will result in increased development time, and once the concentration is shifted from design and development to maintaining the SCM records, the chances of creating an inefficient design arise.

- Potential increase in management effort, difficulties maintaining coordination, and unnecessary generation of requirements, design, test, and system specifications for each selected CI.

Effects of Selecting Too Few CIs

Too few CIs can result in costly logistics and maintenance difficulties, such as these:

- Loss of visibility down to the required level to effect maintenance or modification. For example, if CIs are chosen only at the module level, then maintaining a function in that module will be quite difficult, because finding the subroutine will be difficult when there are not any records in that name. Because the CIs are defined only at the module level, only the modules will be visible.
- Difficulty in effectively managing the changes (for example, managing changes to individual items, which are part of a CI). If there are too few CIs, say, only at the module level, then check-in and check-out and change implementation will be difficult because check-out happens at the CI level, and a lot of unwanted items will also need to be checked out, tested, verified, and checked back in.

Baselines

As the CIs go through their development process, more and more components are developed until the final CIs are available for use. Generally, the life cycle process will first result in a set of requirements, then a design, then code for individual elements of the CI, and then integrated code with test cases and user manuals. The definition of SCM contains the concept of identifying the configuration of each CI at discrete points in time during the life cycle process, and then managing changes to those identified configurations.

The configuration of software at a discrete point in time is known as a baseline. Thus, a baseline is the documentation and software that make up a CI at a given point in its life cycle. It includes the user documentation (if any), the specifications document (or the document that contains the functional and physical characteristics), and software (if any) that make up the CI at a given point. Each baseline serves as a point of departure or reference for the next development stage. Baselines are usually established after each life cycle phase at the completion of the formal review that ends the phase. Thus, we have the requirements baseline, design baseline, and product baseline. Chapter 2, and specifically Figure 2.1, gives an overview of baselines.

Each baseline is subject to configuration control and must be formally updated (using the change management procedures) to reflect approved changes to the CI as it goes through the next development stage. A baseline, together with all approved changes to the baseline, represents the current approved configuration. At the end of a life cycle phase, the previous baseline and all approved changes to it become the new baseline for the next development stage. The term baseline management is often used to describe this control process. Baseline management is the discipline of controlling a series of baselines as they evolve and are then merged into the next

baseline to be defined. The SCM system provides the policies, procedures and tools for exercising baseline management [4].

Baselines are established and placed under CM when it makes good management sense to start controlling subsequent changes to the CIs. When to start baselining is a management decision. Establishing a baseline does not mean that the software development is brought to a halt. It only means that subsequent changes to the CIs will be subject to formal change management procedures. Normally, the first baseline consists of an approved software requirements document and is known as the requirements baseline. The requirements baseline is the initial approved technical documentation for a CI [2]. The requirements baseline contains the details regarding what the system or software must accomplish. Through the process of establishing a baseline, the requirements and other requirements described in the requirements document become the explicit point of departure for software design. The requirements baseline is usually the first established baseline in the SCM process. When the requirements baseline is established at the conclusion of the requirements analysis phase of a project, the formal change control process commences. The requirements baseline is also the basis against which the software is authenticated.

The design baseline is comprised of the initial approved specifications governing the development of CIs that are part of a higher level CI [5]. The design baseline represents the next logical progression from the requirements baseline and represents the link between the design process and the development process. The requirements baseline describes the functions that the system should have. Then, the software system is designed by allocating the various functions to various components or subsystems.

Based on the RDD (part of the requirements baseline), the different functions of the software product are determined, and during the design process these various functions—user interfaces, database operations, error handling, and input data validation—are allocated to the various subsystems and components. This allocation of the different functionality happens during the design phase, and at the end of the design phase the design baseline is established where the components of the system would have their functionality assigned to them. So at the design baseline stage, the functionality, or requirements defined in the requirements baseline, is allotted to CIs in the form of design documentation (high-level and low-level), which is now managed within the design baseline. So at the design baseline stage, the functionality, or requirements defined in the requirements baseline, is allotted to CIs in the form of design documentation (high-level and low-level), which is now managed within the design baseline.

The product baseline corresponds to the completed software product delivered for system integration. The product baseline represents the technical and support documentation established after successful completion of the functional configuration audit and physical configuration audit. The product baseline [6] is the initial approved technical documentation (including source code, object code, and other deliverables) defining a CI during the production, operation, maintenance, and logistic support of its life cycle. The product baseline is sometimes known as the deliverables baseline.

According to Ben-Menachem [4], the process of establishing baselines describes the construction of the aggregates (the software system) from the components (the

CIs). So the baselines also define such things as how the aggregates must be constructed, what tools should be used, which parts are connected and which are not, and what the nature of these interconnections is. As mentioned previously, once the item is tied into a baseline, changes can be made only through the formal change management process. Thus, baselines define the state of a system at a given point in time, and the proper recording of this information is absolutely critical for the success of any project.

CI Selection

A software system is generally split into a number of CIs that are independently developed and tested, and then finally put together at the software system integration level. Each CI essentially becomes an independent entity as far as the SCM system is concerned, and the SCM functions are carried out on each CI. The division of the software into CIs may be contractually specified or may be done during the requirements analysis or preliminary design phase. As a general rule, a CI is established for a separable piece of the software system that can be designed, implemented, and tested independently. Some examples of items that are to be identified include project plan, requirements specifications, design documents, test plans and test data, program source codes, data, object code, executables, EPROMS, media, make files, tools, user documentation, quality manual, and SCM plan.

CI selection is not an easy process as there are no hard and fast rules to it. The selection process should involve all the major stakeholders of the project—including company management, software team leader, SCM administrator or manager, QA representative, testers, maintenance and support team members (if identified), and client representatives.

Checklist for Selection of CIs

When a software project or system is broken down into components (the structural decomposition), we can create a sort of tree structure. The decomposition can be, for example, project-modules-sub modules-programs-functions-link libraries-icons and other small minor components and so on. Decomposition to this detail is neither necessary nor advisable because it will create many problems in managing the CIs and will create too many specifications and documentation. However, if we reduce the number of CIs by limiting the system decomposition to, say, the second level (module level), it will result in poor visibility of the overall design and development requirements.

The number of CIs is determined by the system granularity desired by the SCM system's designers. The granularity decides the level of decomposition of the software system. There are no hard and fast rules here. It varies from project to project and is a decision that has to be made by the SCM system development team.

We have seen that there are no firm rules when choosing CIs, but the following questions can be used as a guide:

- Is the item critical or high-risk or a safety item?
- Is the item to be used in several places?

- Will the item be reused or designated for reuse?
- Is the system already a CI?
- Is the item borrowed from some other project or system?
- Would the item's failure or malfunction adversely affect security, human safety, or the accomplishment of a mission or have significant financial implications?
- Is the item individual, and can it be designed, developed, tested, and maintained as a stand-alone unit?
- Is the item newly developed? For example, a system or subsystem might be developed to add certain requested enhancements.
- Will the item be maintained by diverse groups at multiple locations?
- Does the item incorporate new technologies?
- Is the item purchased off-the-shelf—COTS?
- Is the item supplied or developed by a subcontractor?
- Is the item highly complex, or does it have stringent performance requirements? For instance, does the item have complicated algorithms, or does it need to meet a stringent performance requirement such as having a small footprint?
- Does the item encapsulate interfaces with other software items that currently exist or are provided by other organizations?
- Is the item installed on a different computer platform from other parts of the system?
- Does it interface with other CIs whose configuration is controlled by other entities, for example, a system that interfaces with an off-the-shelf package?
- Is it likely to be subject to modification or upgrading during its service life? Is the item subject to modification at a rate that is much higher than that of the other items? For example, consider an interface that reads data from some external source. Every time the external source changes (assuming that data formats will be subject to frequent changes), the item has to be modified.
- Is there a requirement to know the exact configuration and status of changes made to an item during its service life? This refers to the criticality of the item. Some items in a system are critical or more important than others. So project management will necessarily be more interested in those types of items.
- Is the size of the item manageable? Can it be made by a small but efficient development team in a reasonable time? If not, should it become two items instead of one?

If the majority of these questions can be answered, "No," the item should probably not be a CI. If the majority of questions can be answered, "Yes," the item should probably be a CI. However, there are no definitive rules and no magic formulas to help in CI selection. The bottom line is that selection of CIs is a management decision based on experience and good judgment.

Designation: Naming of CIS

Each software component must be uniquely identified. According to IEEE [3], the identification methods could include naming conventions and version numbers and letters. The identification system or the naming convention should facilitate

the storage, retrieval, tracking, reproduction, and distribution of the CIs. A good naming system will make it possible to understand the relationship between the CIs.

A good naming system uses numbers or alphabets to represent the position of the CI in the hierarchy. For example, an item labeled 1.4 is definitely created after an item with the label 1.2 and before one with the label 1.6. Note that the number is only a part of the name-the time–related part that changes when the item undergoes revisions. The other part of the name, which defines the item, is usually derived from the project or system and the type of the item and the item name. It can be a simple name such as "PGMPAY" indicating that it is the payroll calculation program named PGMPAY or it can be a composite name like "PRJ_MOD PGM_SRC PGMPAY" indicating the project, module, CI type, and name. The decision on the complexity and detail of the names will depend on the size and complexity of the project.

One important thing to remember is that the naming system should be developed in such a way that the derived names do not produce duplicates, because this can create chaos and confusion. For large and complex projects that have thousands of CIs the naming system is usually quite detailed to facilitate easy identification and tracking. As we have seen, the name part of the identification system will remain constant over time for each item, whereas the number part will undergo changes.

CI Description

Software components are described in specifications (i.e., software requirements specifications, software architectural design specifications, software-detailed design specifications, interface control documents, and software product specifications). The description of the component becomes more detailed as the design and development proceeds through the life cycle. The description forms the basis for configuration control and configuration status accounting. The description is also the basis for the configuration audits and reviews, which ensure that the software is complete and verified. The documents or portions of documents that describe each CI must be identified and made part of the CI. I have frequently found a document in use by several different companies (large and small) called the CI description (or specification). Generally, it is not much more than a listing of CIs, by configuration identifier, and an indication of the owner or programmer, next higher CI name (parent), controlling baseline, fit into the hierarchy (CSCI, subsystem, or segment name), and perhaps some other general information. This document forms a handy reference for the CIs.

Acquisition of CIs

The last activity of the configuration identification function is the acquisition of the CIs for configuration control. Following the acquisition of a CI, changes to the item must be formally approved as appropriate for the CI and the baseline involved, as defined in the SCM plan. This means that the CIs—both intermediate and final outputs (e.g., source code, executable, user documentation, design documents,

databases, test plans, test cases, test data, project plan, and SCM plan), and the elements of the environment (such as compilers, operating systems, and tools)—should be acquired and stored in a controlled environment (controlled software libraries) so that they can be retrieved and reproduced when required and access can be restricted. A software library is a controlled collection of software and related documentation designed to aid in software development, use, and maintenance [4]. IEEE [6] specifies that for each such library the format, location, documentation requirements, receiving and inspection requirements, and access control procedures must be specified. Once the CIs are acquired and placed in the controlled library, as we have seen, the configuration control procedures will apply to them.

Summary

Configuration identification is the process of selecting the CIs for a system and recording their functional and physical characteristics in technical documentation. We have seen why it is important to select the CIs, how to select them, and how to establish baselines. Configuration identification is one of the important phases of SCM, because it decides the level of granularity or detail to which SCM will be performed in a project.

References

[1] Alain Abran, A., and J. W. Moore (eds.), *SWEBOK: Guide to the Software Engineering Body of Knowledge (Trial Version)*, Los Alamitos, California: IEEE Computer Society, 2001.

[2] *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610–1990), IEEE Standards Collection (Software Engineering)*, Piscataway, NJ: IEEE, 1997.

[3] EIA, *National Consensus Standard for Configuration Management (EIA-649)*, Arlington, VA: Electronics Industries Alliance, 1998.

[4] Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

[5] Ben-Menachem, M., *Software Configuration Guidebook*, London: McGraw-Hill International, 1994.

[6] *IEEE Standard for Software Configuration Management Plans (IEEE Std-828–2012)*, Piscataway, NJ: IEEE, 2012.

Selected Bibliography

Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, 1992.

Dart, S., "Concepts in Configuration Management Systems," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1994.

Feiler, H. P., "Software Configuration Management: Advances in Software Development Environments," Technical Paper, Software Engineering Institute, Carnegie-Mellon University, 1990.

IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition), Piscataway, NJ: IEEE, 2003.

Magnusson, B. (ed.), *System Configuration Management: ECOOP'98 SCM-8 Symp. Proc.*, Berlin: Springer-Verlag, 1998.

Quality Management-Guidelines for Configuration Management, ISO 10007:1995(E), Geneva, International Standards Organization, 1995.

CHAPTER 7

Configuration Control

Introduction

IEEE [1] defines *configuration control* as an element of CM, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to CIs after formal establishment of their configuration identification. So once the CIs of a project or system have been identified, the next step is to bring in some degree of control. Restrictions have to be implemented, and rules regarding who can do what to these CIs have to be formulated. This aspect of SCM is handled by the configuration control function or the "change management and control system."

Changes to all items identified in the configuration identification phase—CIs—should be controlled. To properly control change, procedures have to be established, guidelines have to be implemented, roles and authorities have to be defined, and all workflow processes of the change management system—including change identification, change requisition, change approval or disapproval, change implementation, and testing—have to be designed, documented, and implemented. How these procedures and guidelines apply to the different CIs such as source code, documents, specifications, third-party software, and subcontracted items also has to be established. These activities fall under the purview of configuration control.

Of the SCM functions, configuration control is the one that is performed most often. Configuration identification, as we saw in Chapter 6, is done only once at the beginning of the SCM implementation. Status accounting, which is the recording and reporting of the SCM activities, is done on a regular basis. Configuration audits are performed when a configuration is complete or before a system is released. However, the configuration control activities have to be done whenever a change request (CR) is initiated. Requests for changes will be quite frequent during the software development phase and after the system is released. Requests for new features, functional enhancements, and bug and defect reports can all initiate a change.

The different activities of configuration control lend themselves to automation. For example, all activities from change initiation to change disposition can be easily automated. Thus, configuration control is one SCM function that can be automated very efficiently and effectively. In fact, it is imperative that one use some sort of change management tool, because except in the case of very small single-person projects, change management and control is too repetitive and monotonous, and hence, more prone to error and not worth the effort for manual processing. Many software tools are available—covering almost all available platforms and development environments—to automate configuration control. In fact, it was one of the

first functions of CM that was automated, which is evident from the number of change management tools available on the market.

Configuration control is not an easy task. It involves a lot of people and a lot of procedures, making it difficult to manage. The configuration control activities will increase as the project evolves, because more and more items will undergo change, more and more people will be inducted, requirements will change, new modules and subsystems will be added, and different versions will have to be maintained—among other challenges. However, if designed intelligently, planned properly, and supported well by a good software tool it can be an easy, if not exciting, task.

Change

Change is one of the most fundamental characteristics in any software development process. All phases of the software development process from requirements analysis to production or maintenance are always subject to change. Making changes to software is easy. In fact, it is one of best features of software—that it can be changed at will. However, if changes are made at will, without any proper planning, chaos will result. Making changes is easy; managing those changes—the uncontrolled changes—is not, because there is no way of knowing what was changed and hence what to manage.

Software development is a continuously evolving process. You cannot freeze one phase of software development and then go to the next phase. Even though early development models like the waterfall model were developed based on the compartmentalization of the various phases, the real-life situation is quite different. You cannot freeze analysis and go to design and freeze design and then go to development. A great deal of overlap occurs between these phases. This is because software development is a complex process that involves many variables, all of which can change. Changes to the requirements drive the design, and the design changes affect the code. Testing then uncovers problems that result in further changes, which might force us to return to the requirements phase. So change is something that cannot be avoided. Managing the change process is a complex but essential task.

Proposing Changes to the Customer

In some instances, a change affects the customer's agreement with the developer. These changes (sometimes known as major changes) can affect the terms and conditions of a contract or purchase order related to cost, delivery schedules, or other milestone events. Other changes can affect the statement of work, specifications, or other requirement documents to which the customer has asked the developer to adhere. Thus, when these types of changes occur, a CR is prepared and submitted to the customer for approval.

Incorporation of such proposed changes cannot be implemented until the customer has given approval to do so. The procedure for processing these changes is the same as the normal change management procedures (Figure 7.1). A point to remember is that once a major change is approved, it is actually an amendment to

the developer's contract. In addition, the customer normally pays for the cost of preparing the changes and incorporating them into the system and conducting the prescribed tests. Such changes may also require reidentification of the change units or modules and may, in addition, affect those of which they are a part.

Deviations and Waivers

In addition to changes that may affect the functionality of the software product, developers may also run into situations where they must deviate from the prescribed specification due to a temporary inability to meet a given requirement. The constraints imposed on a software development effort or the specifications produced during the development activities might contain provisions that cannot be satisfied at the designated point in the life cycle. The customer may approve such deviations up to an agreed on point in time or in some instance may approve a permanent deviation without requiring a major change to be submitted.

A deviation is an authorization to depart from a provision prior to the development of the item. A waiver is an authorization to use an item, following its development, that departs from the provision in some way. In these cases, a formal process is used for gaining approval for deviations to, or waivers of, the provisions [2].

Waivers may also be granted to cover temporary problems in meeting specifications or contract requirements. Instances include delivery of the product without having completed all the prescribed testing or delivery of the product without certain units or modules included, due to a delaying action. A waiver, however, has a specified time factor that must be met in order to satisfy contractual or agreed on requirements. The procedure to be followed when requesting for authorized deviations and waivers are normally specified in the contract or agreement.

Change and Configuration Control

Change is inevitable during the software development life cycle. Changes to software come from both external and internal sources. External changes originate from users, from evolution of operational environments, and from improvements in technology, among other factors. Internal changes come from such impetuses as improved designs and methods, incremental development, and correction of errors. A properly implemented configuration control process is the project manager's best friend and provides potential salvation when coping with change.

Configuration control (or change management and control) is thus the process of evaluating, coordinating, and deciding on the disposition of proposed changes to the CIs and implementing the approved changes to baselined software and associated documentation and data. The change control process ensures that changes that have been initiated are classified and evaluated and approved or disapproved and that those that are approved are implemented, documented, tested, verified, and incorporated into a new baseline.

Configuration control is the set of techniques used to ensure that the components in a system achieve and maintain a definite structure (where the relationships between

the components are established) throughout the system life cycle. To this end, change management and control provide the necessary procedures, documentation, and organizational structure to make sure that all items identified in the configuration identification phase, the details of the changes made to them, and other related information are available to all who need to see it (or have the necessary authority) throughout the system life cycle. In other words, configuration control provides the necessary mechanism to orchestrate change—but in a controlled manner.

Problems of Uncontrolled Change

We have seen that uncontrolled or unmanaged change can create problems serious enough to create project failures. In its mildest forms, these changes can create confusion and chaos. Change management and control solves the four most common (and dangerous) software development and maintenance problems: communications breakdown, shared data, multiple maintenance, and simultaneous update problems.

In any development environment, the same code, say a program, function, or subroutine, is often shared by different programmers. This sharing of common items—source code or data or documentation—reduces development costs because it avoids the problem of reinventing the wheel. If a function or a program or a component library that suits one's needs is already available, then it is prudent to use it, rather than coding it again. Similarly, in the case of documentation, such as an RDD or SDD, the entire project uses the same document. So, what is wrong with sharing data, code, or documents? As long as nobody makes any changes to these shared items, there are no problems. However, if changes are made to any shared item without a proper control mechanism, trouble can arise, as detailed in Chapter 3.

In the case of a properly implemented change management system, all changes made to the components of a software system are made after proper analysis and review. Because changes can be made only with proper authorization, and since the authorization is done by a separate entity that is responsible for managing the changes to all the items, the chances of effort getting duplicated or two people solving the same problem in isolation do not occur. Also, the problem of one person overwriting another person's efforts does not occur, because the changes are made to items that are stored in a controlled environment, where records of who is making changes to what items are kept. Accordingly, if a person is making changes on an item, that fact is known to all the people in the project. Also, the information regarding a change is reported to all concerned. Thus, a good change management system solves the abovementioned software problems and can bring discipline into the development process and improve development productivity, because a lot of time that would otherwise be spent on debugging and reworking can be saved. The following sections describe how this is accomplished.

Configuration Control

We have seen the dangers and problems of unmanaged and uncontrolled changes. So how do we to avoid them? We should have a good change management and

control system. The system should define a process and the necessary procedures to ensure that all events—from the identification of a change to its implementation and baselining—are done in a systematic, scientific, and efficient manner (i.e., following SCM principles).

To make this happen, procedures should be established for requesting a change once the need has been identified and people authorized to decide whether the requested change needs to be implemented. Once the decision to implement the change has been made, a mechanism should be in place for analyzing which other resources are affected by the proposed change and then assigning the task of effecting the change to the resource (and if necessary to any impacted resources). The necessary facilities to test, verify, and validate the changed resources also need to be in place; in other words, the changed function or program needs to be tested, verified, and approved so that it can be incorporated or promoted as the new version.

An orderly change process is necessary to ensure that only approved changes are implemented into any baselined document or software. Figure 7.1 shows a simple overview of the change management and control process. The steps mentioned here are very generic and will vary from one company to another and even from one project to another.

The steps within the overall process can be grouped into the following categories:

- Change initiation;
- Change classification;
- Change evaluation or change analysis;
- Change disposition;
- Change implementation;
- Change verification;
- Baseline change control.

These seven steps in change control are individually discussed in the following sections. We will first look at how configuration control is done manually. As a minimum, it is suggested that the following data elements always be included (as applicable) in any change control communication:

- Project name, date, requestor name, and priority (high, medium, or low);
- Name, number, and description of element(s) needing change;
- Description of change(s) required or made;
- Suggested fixes or fixes made with supporting data if needed;
- Disposition of change by a review board or person;
- Approval signatures of review board or person;
- Date of incorporation;
- Date of verification of the incorporated changes.

If one is using a change management tool or SCM tool, then most of these processes will be done automatically. For example, processes such as the change requisition, verification of the details, assignment of CR numbers, intimation of the CR evaluators, intimation of the change control board members, voting on the CR, and informing the result could be done automatically. Project managers should be

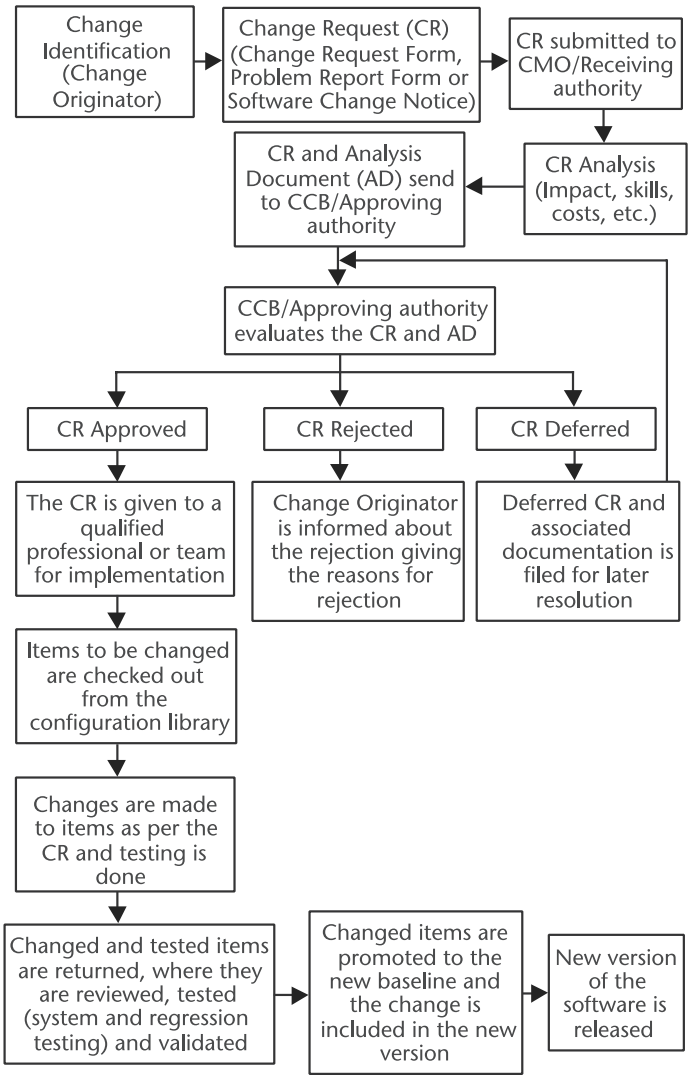


Figure 7.1 Overview of change management and control process.

careful, however, when selecting a tool that the one under consideration fits the job at hand and that the project team does not have to change form, fit, and function of the job to fit the tool. There are any number of consultants and tool reference guides available to help a project or firm, select the type of tools that will enhance productivity and maintain good change control. This topic is detailed in Chapter 15.

Change Initiation

Requests for change to software and documents come from many sources. A CR can be submitted by the developer, a member of the QA team, a reviewer, or a user. Each project should set up a CR form for documenting the proposed change and its disposition. Sometimes a CR is also called a problem report (PR) or a specification change notice (SCN).

Problem reports, which we will discuss later in the chapter, are a special kind of CR where the cause of the change is a defect or bug in the system. However, problem reports necessitate change, so the procedures for resolving problem reports and for requesting an enhancement or a new product feature are the same. Figure 7.2 shows a sample CR form. The sample contains the basic information that should be included in a CR/PR/SCN form; however, the actual form for a particular project must correspond to the planned SCM process. Note that electronic forms, containing the same information, are being used increasingly as direct interfaces to SCM support tools.

Each project should also name an individual—the CM officer (CMO) or a member of the SCM team—to receive the CR form, assign it a tracking number, and classification, and route it for processing. This person receives the CR and reviews it for clarity and completeness. If the CR is not complete, it is returned to the originator. Once complete, the CR is assigned a unique identifier for tracking purposes, and the information about the CR is recorded in the CR tracking database or files.

In an automated environment the elements of the CR form are available online and simply require the originator to bring the form up and enter the necessary data.

CHANGE REQUEST

CR No.:.....

Analysis Document No.:.....

System/Project..... Item to be changed:.....

Classification: Enhancement / Bug Fixing / Other:.....

Priority: Immediate / Urgent / As soon as possible / Desirable

Change Description:

Status		Date	By	Remarks
Initiated				
Received				
Analyzed				
Action (A / R / D)*				
Assigned				
Check-out				
Modified & Tested				
Reviewed				
Approved				
Check-in				
Baselined				

* A - Approved, R - Rejected D - Deferred

Remarks:

Figure 7.2 Sample CR form.

There are many ways to make this easier than the manual entry, especially when redlining the changes to be made on a source or object listing.

Change Classification

Changes to software and associated documentation are classified according to the impact of the change and the approval authority needed. Depending on the criticality, impact, and cost involved, there will be a hierarchy of people who can approve the changes. At the top of the hierarchy is the CCB, which is detailed later in this chapter. Major changes need the approval of the CCB, whereas minor changes can be done with the approval of the project manager or development manager. This is usually done to speed up (fast-track) the change management process. The exact mechanism of the change classification and the approval should be defined in the SCM plan. (See Chapter 13 for more on SCM plans.)

The changes are classified into different categories with different priorities. Classification methods can be based on such factors as severity, importance, impact, or cost involved. For example, a CR for fixing a bug that could result in system failure will have higher priority than a request for a cosmetic change. A functional enhancement request that comes from a user may not be in the same category as a CR from a member of the development team. However, the classification criteria (how to classify a change) should be spelled out very clearly in the SCM plan.

The individual who proposes the change may suggest a classification for that change. The CMO or the receiving authority reviews suggested classes and assigns a working or tentative classification. After assessment of the impact of the CR, the CCB or the approving authority will assign the final class.

Change Evaluation/Analysis

One important aspect of the configuration control process is that it provides adequate analysis of changes in terms of impact to system functionality, interfaces, utility, cost, schedule, and contractual requirements. Each change should also be analyzed for impact on software safety, reliability, maintainability, transportability, and efficiency. The project CMO routes the CR to the software engineering staff for evaluation.

In some cases, project procedures require that the CR be screened before it is analyzed. Some CRs will not have any chance of approval due to considerations (e.g., costs or schedules) of which the change initiator may not be aware. In some cases, management may decide not to take any action in the case of changes that fall into some category or meet some predefined criteria. This information might not be or need not be communicated to all the people involved in the project. So when such CRs—the CRs that do not have any chance of approval are submitted—they will get rejected in the pre-evaluation screening itself. This approach saves the cost of analysis for changes that do not have any chance of approval.

The analysis produces documentation (like that shown in Figure 7.3), which describes the changes that will have to be made to implement the CR, the CIs and documents that will have to be changed, and the resources required to effect the change. This documentation becomes part of the change package, along with the CR. After completion of the analysis, the change package is sent to the CCB.

CHANGE ANALYSIS DOCUMENT

No.:.....

CR No.:.....

Date:.....

System/Project:..... Item to be analyzed.....

Analyzed By:.....

Implementation Alternatives:

Items Affected

Item ID	Item Description	Version No	Nature of Change

Estimated Effort:.....

Impact on Schedule:.....

Impact on Cost:.....

Recommendation:

Figure 7.3 Sample change analysis document.

Change Disposition

Disposition of changes to baselined items are usually done by a CCB. The CCB evaluates the desirability of a change versus the cost of the change, as described in the documentation of the analysis. The CCB may approve, disapprove, or defer a CR. Sometimes the CCB may have to request more information and additional analysis.

Items for which decisions have been made are sent to the CMO for action. Rejected items are sent to the originator along with the CCB's rationale for rejection. CRs needing further analysis are sent back to the analysis group with the CCB's questions attached. Deferred CRs are filed, to be sent back to the board at the proper time.

Remember that the CCB may not be the change approving/disapproving authority in all cases. In some cases the project leader, the CMO, or any other designated person could make the decision. The exact mechanism of change disposition varies from one organization to another and will be usually documented in the SCM plan.

The CMO sends approved items to the development team. The CMO also prepares and distributes the meeting minutes and records the current status of the CR. This information is added to the tracking database or recorded in files.

Today, with the use of change management tools, physical CCB meetings are rare. In today's development environment, e-mail or some other messaging system connects everybody in the organization. So it is possible to hold CCB meetings without the CCB members actually meeting. The CRs and the necessary information (such as evaluation reports and impact analysis reports) can be sent electronically to all CCB members, and the CCB members can convey their responses in the same way. Thus, today it is possible to hold virtual CCB meetings and have online voting on CRs. The SCM tools make change disposition and management an easy task.

Change Implementation

Approved CRs are either used directly as a change authorization form or result in a change directive being prepared by the CMO. In either case, approval results in the issuance of instructions that authorize and direct the implementation of the change in the software and associated documentation.

The development team schedules the resources to make the change. It must get official copies of the baselined component to be changed from the program library. For code changes, design has to be developed, code has to be written, testing has to be done, and the correctness of the change has to be verified. Moreover, the associated documentation has to be revised to reflect the change. Once the change has been made and local testing completed, the revised component and documents are returned to the control of the program library. After verification, the new version takes its place in the sequence of baselines.

Change Verification

The implemented changes, which have been tested at the unit level, must be verified at the system level. This may require the rerun of tests specified in the test plan or the development of additional test plans. Regression testing will usually have to be included in the test to ensure that errors have not been introduced in existing functions by the change. Once the verification is complete, the reviewing team submits evidence of it to the program library, which will then accept the changed items for inclusion in the SCM controlled files that make up the new version of the baseline.

After the successful implementation and testing of the change described in the CR, the CMO will record the occurrence of this process into the CR tracking database or files. Also, a change history (or patch history) is maintained. The change history is a recording of the events that occurred to an item from the state before change to the one after. The details to be incorporated include (but are not limited to) name of the originator and receiving authority, date received, name of the individual who performed the analysis, date of analysis, approving authority's name, date, names of the persons who effected the change, testing, review and audit, reasons for change, and a short description of change.

If an SCM or change management tool is used, then such steps as the process of recording the change implementation information and the task of changing the status of the CR do not have to be done manually. All of these activities will be taken care of by the tool. As mentioned before, the tools capture all the information

as the events are happening and will record them automatically. So, details such as when the change was initiated, when it was evaluated, when it was reviewed, who initiated the change, who reviewed it, who approved it, when the implementation started, when it was finished, and who performed the implementation will automatically be captured by the tool. Accordingly, in a project where SCM tools are used the abovementioned activities (the activities that are performed by the CMO or SCM team members in a manual SCM system) are done automatically and without human intervention. However, all these features—the complete automation of the change control process—are available only in the more advanced and sophisticated SCM tools.

Baseline Change Control

Changes to software are not complete until the code and data changes have been implemented and tested, the changes to associated documentation have been made, and all of the changes have been verified. To minimize the number of versions and the frequency of delivery of software components, changes to software are usually grouped into releases. Product release is the act of making a product available to its intended customers [3]. Each release contains software and documentation that has been tested and controlled as a total software system.

There are other reasons for product releases. One would be to satisfy a customer by customizing a software system to meet the specific needs of that customer. This is called a customer-specific release. For properly incorporating emergency fixes (a fix that was done without following any change management procedures due to the urgency of the problem or situation), a release might be made after the emergency fix has been properly incorporated. Alpha and beta releases are also used for alpha and beta testing.

Companies also do major and minor releases. Major releases are done when there is a significant increase in the product's functionality, whereas minor releases are done when the release is to correct a bug or fault in the program or system. The decision on the when and how of the releases is usually made by the CCB, because it is the ultimate authority for making decisions about configuration control and is represented by all functions of the organization.

File-Based versus Change-Based Change Management

In a file-based change management system, to make a change the change initiator identifies the file he or she wants to change and initiates the change management process. The CR is then analyzed, the impacted files are identified during the CR evaluation phase, and the decision to approve or reject the CR is made. If the CR is approved then the file (or files if more than one file is impacted) is checked out and the necessary changes are made to it. Then the file is tested, verified, and checked in. Thus, if there is more than one file for the same CR then they are not associated with one another except for what is recorded in the evaluation report.

The major drawback of the file-based system is that it fails to capture the relationships between the items that are changed due to a CR. In real life, a typical change

is rarely limited to a single file; in most cases more than one file needs to be changed to implement a CR. However, the problem with a file-based change management system is that once the files are checked in there is no way to determine which files were modified as a result of a particular CR. Yes, the person who has implemented the change might know or there might be some informal records somewhere, but there are no formal methods to track all the files that were modified in response to a single change or CR. This creates a lot of problems, because people often forget the details of all the files they changed and often forget to include some of them during the system building, resulting in build failures.

To avoid the drawbacks of file-based change management, SCM practitioners started to use change-based change management. In this system, all the files required to perform a task or to implement a CR are considered to be a single entity. Here we are tracking logical changes rather than individual file changes. However, the technology of making all the files of a CR into a single logical unit is not new. Some mainframe systems tracked changes in this manner as early as the 1970s, and companies such as IBM, Control Data Corporation, Unisys, and Tandem have used logical change-based software tracking systems for years [4]. In 1983, SMDS (now True Software) released Aide-de-Camp as the first commercial SCM system that tracked logical changes rather than physical file changes [5].

Since then, many commercial SCM systems have added the ability to track logical changes rather than individual file changes, including Synergy/CM from Telelogic AB, AllFusion Harvest CM from Computer Associates, Dimensions from Merant, and ClearCase from Rational. SCM tools like Merant's PVCS and StarBase's StarTeam have the ability to mark a source code change with the corresponding defect report or enhancement request.

The method of tracking software by units of logical change is a more logical and practical model, because the items that are changed because of a single CR are logically linked. They are checked out together, they are tested together, they are reviewed and approved as a group, and they are checked in and promoted together. According to Weber [6], not all SCM systems use the same name for the logical unit of change. For example, ADC/Pro uses the term "change set," AllFusion Harvest CM uses "package," Synergy/CM and Dimensions use "task," CIearCase uses "activity," PCMS uses "work package," and StarTeam uses "subproject."

Furthermore, not all SCM systems implement the ability to track changes in the same way. Two very different implementations have emerged—change sets and change packages. Systems that treat a logical change as the individual lines of code typically refer to the unit of change as a change set. Systems that treat a logical change as the set of file versions that contain the code changes are called change packages. For a detailed discussion of change sets and change packages, readers should refer to the following texts:

- Burrows, C., S. Dart, and G. W. George, *Ovum Evaluates: Software Configuration Management*, London: Ovum Limited, 1996.
- Cagan, M., and D. W. Weber, "Task-Based Software Configuration Management: Support for 'Change Sets' in Continuus/CM," Technical Report, Continuus Software Corporation, 1996.

- Weber, D. W., "Change Sets Versus Change Packages: Comparing Implementation of Change-Based SCM," *Proc. 7th Software Configuration Management Conf. (SCM7)*, Boston, MA, May 1997, pp. 25–35.
- Weber, D. W., "Change-based SCM Is Where We're Going," Technical Report, Continuus Software Corporation, 1997.

Escalation and Notification

Escalation can be defined as the process of increasing the intensity or magnitude of an issue. In the change management process, there are times when issues need escalation. For example, consider a CR for which the evaluation report was forwarded to all CCB members for their decision. If a CCB member has not conveyed a decision within the specified time period, then the person has to be reminded about it. However, if, even after the reminder, nothing happens, then the issue has to be brought to the attention of the senior member of the CCB, so that necessary corrective action can be initiated.

The escalation process is equally applicable for most of the change management processes such as change evaluations, impact analysis, and change implementation. Also, we have seen that the CRs can be accepted, rejected, or deferred. In the case of deferred CRs, a time period can be set after which it has to be revisited. So once the specified time is over, the CR is again reviewed. This process of keeping track of the deferred CRs and then bringing them back for review is another form of escalation.

Today's change management tools are capable of performing problem escalation and notification automatically based on predefined rules and criteria. For example, the change management tools could be programmed to escalate an issue (like failure to convey the decision on a CR) after a specified number of days.

Multiple levels of escalation are also possible. For example, if the CCB member fails to respond to the reminders, then the issue could be escalated to his or her superior, and if there is still no action after a specified period, the next person in the organizational hierarchy could be informed about the issue. Here, such factors as the levels of escalation, the time period before escalation, and the people who are to be informed can be predefined, and the tools will do the rest. This is an important aspect that will improve the efficiency and productivity of the SCM team, since it will not have to keep track of each and every CR; the tools will automatically perform the necessary actions when something is not happening according to the rules and schedules.

Emergency Fixes

Some CRs or problem reports need immediate action and will not allow enough time to follow all change management and control procedures. For example, an emergency request from a client or a distress call from a customer cannot wait for steps such as the change evaluation, CCB meeting, or change disposition. Efforts to correct these difficulties are referred to as emergency fixes. These are not change

management processes in the conventional sense, because the sole focus of an emergency fix is to resolve the customer's difficulties right away. The most important distinction is that these emergency fixes invalidate the version of the components that they fix, because they are temporary measures taken when there is not enough time or resources to process them in the proper manner. When time permits, these emergency fixes will be taken through the proper change management procedures and the required tasks will be completed.

Problem Reporting and Tracking

We have seen how the change management and control process works, starting with the initiation of the CR and the subsequent processing to effect the change. A CR can result from many things. It can be the result of a user needing a new feature, it can be the result of some enhancements of the existing functionality, or it can be due to an anomaly in the software system. An anomaly is any condition that deviates from expectations based on such information as that contained in the requirements specifications, design documents, or user documents, or someone's perceptions or experiences. Anomalies may be found during, but not limited to, the review, test, analysis, compilation, or use of software products or applicable documentation [7]. In common usage, the terms error, fault, flaw, gripe, glitch, defect, problem, and bug are used to express the same meaning. In this section we deal with PRs or software PRs (SPRs). The SPR or PR is a type of CR—a request that is the result of an anomaly in the system.

Problem Reports and CRs

An SPR usually will get more and quicker attention than a CR because it is the result of a problem that has to be fixed. It is not some cosmetic change that can wait. Also, a single SPR can create or result in more than one CR. This is because a problem or bug (for example, navigation not working properly) can be the result of faults in two different subsystems that require different skills to fix and thus need two different persons or teams to do the job. Moreover, it might be better to keep the two subsystems separate if there are no direct relationships or dependencies between the two. Accordingly, it is quite possible that a PR or SPR can initiate more than one CR.

We saw that SPRs can result in one or more CRs. In the author's opinion, the disposition of a fault or PR should follow the same process as that of the enhancement requests once the CRs associated with the PR have been created. In other words, the processing of the PR and the enhancement request become the same. In the case of a PR, however, before the creation of the CR and after the change management process is set into motion, some activities need to done. These activities are intended to prevent the same type of mistakes from recurring and to create a knowledge base of the anomalies. Figure 7.4 shows the problem reporting and tracking process.

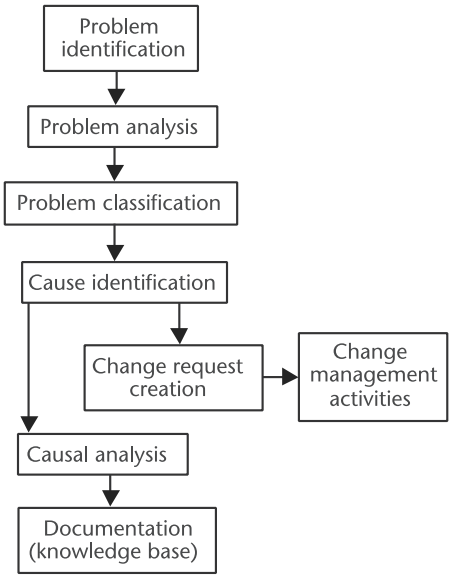


Figure 7.4 Problem reporting and tracking process.

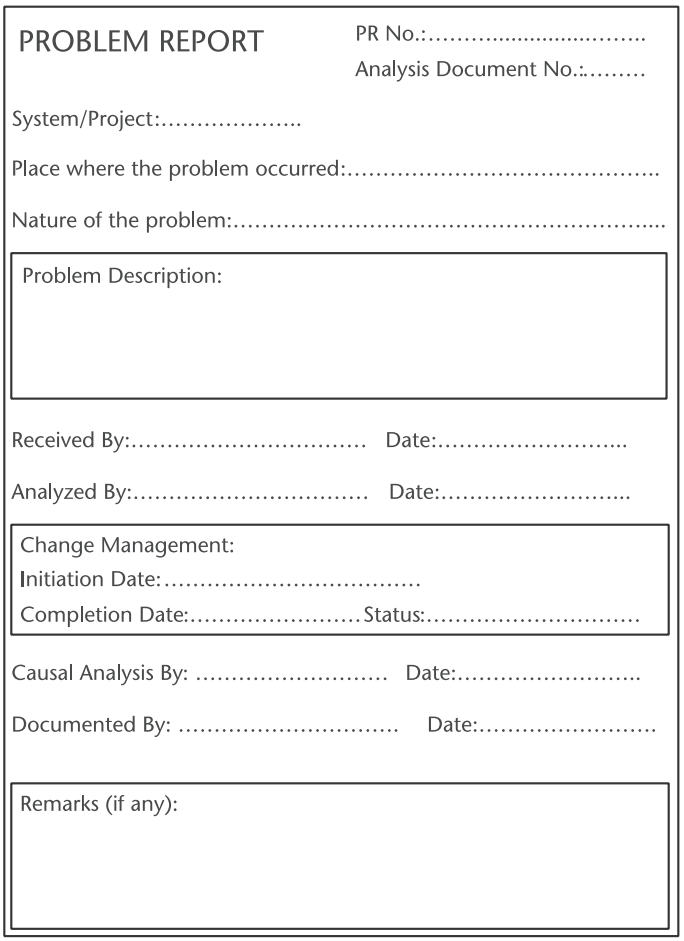
Problem Identification

We know that a problem, bug, or defect can go undetected and can remain in the software system. It is not possible to say that a software system is 100% defect-free. Nevertheless, we need to try to reduce the number of defects in a system and, more importantly, reduce the number of critical defects. We know about the existence of a defect when, for example, something goes wrong, the system starts misbehaving, the performance of the system is not what it is supposed to be, or the system stops performing.

Once the problem is identified, it should be reported and fixed, and the fixed version should be reviewed, approved, and baselined. So, the first step after the detection of an anomaly or defect is to report it. Figure 7.5 shows a PR form. PRs are usually handled by the SCM team, so the PR is also received by the CMO or a representative of the SCM team. Once the PR is received, it is checked for clarity and completeness and if the necessary details are specified, the PR is assigned a PR number. This number serves as the identifier for the PR.

Once the problem is known and the report is received, the report is given to qualified professionals (people from the project team or QA department who have the necessary technical knowledge of the system and the problem analysis methodologies) for analysis. These people will analyze the problem with the objective of determining the severity of the problem; its nature; its impact; the cause, the category; its place of origin; the items affected; and the cost, time, and skills required for fixing the problem. The analysis will also classify the defect (based on its seriousness and impact) and will create the CRs for fixing the problem. As mentioned previously, a PR can result in more than one CR.

The analysis team will report their findings in the problem analysis document. Figure 7.6 shows a sample problem analysis document. This document forms the


PROBLEM REPORT

PR No.:……

Analysis Document No.:……

System/Project:……..

Place where the problem occurred:……..

Nature of the problem:……..

Problem Description:

Received By:…… Date:……...

Analyzed By:…… Date:……...

Change Management:

Initiation Date:……

Completion Date:……Status:……

Causal Analysis By: …… Date:……..

Documented By: …… Date:……

Remarks (if any):

Figure 7.5 Sample PR form.

basis of causal analysis, which is discussed later in this chapter. The CRs that are created as a result of the problem analysis are processed as discussed in the previous sections.

Defect Classification

The classification of a defect is dependent on the phase in which it occurs. The following is a general classification of defects during the various phases of a project.

Requirements Analysis

- *Incorrect requirements*. This occurs when a requirement or part of it is incorrect. This may result from a misunderstanding of the user expectations.
- *Undesirable requirements*. The requirement stated is correct but not desirable due to technical feasibility, design, or implementation cost considerations.

PROBLEM ANALYSIS DOCUMENT

No.:……..…….....
PR No.:……..…….....
Date:……..…….....

System/Project……..…….....
Analyzed By:……..…….....
Severity: Critical / Fatal / Non-fatal / Cosmetic
Classification:……..…….....

Cause of the problem:

Items Affected

Item ID	Item Description	Version No	Nature of Change

Impact on Cost:……..……..... Time:……..…….....
Skills required in fixing the problem:……..…….....

Implementation Alternatives:

Recommendation:

CR(s) created:……..…….....

Figure 7.6 Sample problem analysis report.

- *Requirements not needed*. The user does not need the stated functionality or feature. Adding this requirement does not significantly increase the utility of the project.
- *Inconsistent requirements*. The requirement contradicts some other requirement.
- *Ambiguous or incomplete requirements*. The requirement or part of it is ambiguous. It is not possible to implement the stated requirement.
- *Unreasonable requirements*. The requirement cannot be implemented due to cost, hardware, or software considerations.
- *Standards violation*. The standards set for analysis have not been followed.

Design Phase

Design defects may relate to data definition, the user interface, the module interface, or processing logic. Each of these may be incorrect, incomplete, inconsistent, inefficient, or undesirable, or they may violate standards.

Coding and Testing Phase

Coding and testing defects may relate to such factors as logic, boundary conditions, exception handling, performance, documentation, and standards violations.

The above classifications are very general, and depending on the nature of the project and degree of detail required, a defect classification system should be designed for each project.

Defect Severity

Severity of a defect is a measure of the impact of the defect. During the analysis, design, and coding phases the defects might be classified as follows:

- Major defects, which have substantial impact on several processes or subsystems. The correction activity involves changing the design of more than one process or subsystem.
- Minor defects that impact only one process or subsystem. The correction activity will be local to that process or subsystem.
- Suggestions toward improvements.

During the testing phase the defects may be classified as follows:

- Critical. These are errors that cause system failure.
- Fatal. These are fatal errors that result in erroneous output.
- Nonfatal. These are errors that are not fatal but will affect the performance or smooth functioning of the system.
- Cosmetic. Minor errors like cryptic error messages or typos in messages, screens, or user documentation.

During problem analysis, the analysis team classifies the defect, decides on its severity, and records these findings in the problem analysis document.

Defect Prevention

The primary objective of the problem report is to identify the fault and fix it. The problem analysis and CR generation and change management process achieve this. The secondary objective (maybe one that is more critical in a long-term perspective) is to prevent faults from recurring. This area of problem identification and tracking is called defect prevention.

One of the main methods of defect prevention is causal analysis. The other method is the creation of a knowledge base that contains the classified and categorized defects, so that a programmer or designer can browse the knowledge base before he or she starts the analysis, design, or development, to be forewarned of the problems that could occur.

Causal Analysis

The objective of causal analysis is to analyze defects and problems to determine and record the cause and initiate corrective actions so that the defects will not occur again. The primary document or the basis of the causal analysis is the problem analysis document. This document contains information such as the causes of the defects, where they occurred, and their severity.

By studying the analysis reports, the person doing the causal analysis will be able to come up with cause patterns. For example, some of the causes for the defects are insufficient input during the analysis stage, inadequate standards, inadequate skill levels, lack of training, lack of documentation, lack of communication, oversight, and inappropriate methodology or tools. For example, in a project if the causal analysis reveals that the reason for most of the defects is lack of knowledge of a tool that is used to generate the code, the team members can be given training on the tool so that the problem can be prevented. Thus, causal analysis plays an important role in defect prevention.

Defect Knowledge Base and Help Desks

When a problem analysis document is submitted, it should result in the creation of CRs and problems being fixed. It will also form the basis of causal analysis. Its contents should also find their way into the defect knowledge base—a knowledge base that stores defects in an organized way, classified and categorized. This knowledge base should have a search facility where one can search for defects by such aspects as category, phase of origin, cause, and severity. Details of the defects such as the project, the defect description, the cause, and the solution should be in the knowledge base.

This knowledge base will be tremendously valuable because it will serve as a road map and guidebook for analysts, designers, programmers, and people who do the testing and maintenance. For analysts, designers, and programmers, it will serve as a guide, telling them, for example, what to do, what to avoid, and what mistakes can happen. The people who do the testing can create better test cases and test data if they know about the defects that escaped testing and how that happened. People who do the problem fixing will find similar problems and can see how they were fixed. This information will be very useful for people who are managing the help desks and for the technical support team. In addition, as new problems get added to the system, it will become more and more comprehensive, and its usefulness will increase.

CCB

We have seen that once the CIs are identified, acquired, and baselined, they come under the purview of configuration control, and formal change control procedures come into effect. This means that once the items are brought into the SCM system, the changes to it are done through a formal change management process. Previous

sections have described the exact mechanism of this process. Since the SCM is one of the essential means of communication, changes to agreed-on baselines or points of departure should be reviewed and approved to ensure that the integrity of a baseline has not been altered by a given change. In addition, such a review and approval should be made for all internal changes so that communication among the development team members can be maintained.

We have also learned that there should be a body for making decisions such as what changes should be made to a CI and which CRs should be rejected This approving authority is known as the CCB or the change control authority (CCA). The IEEE [1] defines the CCB as a group of people responsible for evaluating and approving or disapproving proposed changes to CIs and for ensuring implementation of approved changes. This also ensures that all proposed changes receive a technical analysis and review and that they are documented for tracking and auditing purposes. The board also has final responsibility for release management (e.g., establishing new baselines).

The basic tasks of the CCB are to declare baselines on CIs (e.g., promotions and releases; to review changes to baselined CIs; and to approve, disapprove, or defer their implementation. The above is a short but extremely important task list. The CCB must have a stranglehold on the project. Nothing can be changed without its approval except in the case of emergency fixes. For this reason, board members must be chosen carefully.

CCB Composition

The name change control board has the connotation of being a bureaucratic setup with many people. The composition of the CCB can vary anywhere from a single person to a highly structured and very formal setup with many people. For example, in small projects, the project leader alone will perform all the functions of the CCB, but in the case of large projects like a defense project, there will be a highly structured and formal CCB setup with well-defined procedures.

Factors such as the composition, the nature of functioning (formal or informal), and the number of people in the CCB depend on the complexity, size, and nature of the project. In some very large projects there can be multiple levels or hierarchies of CCBs (CCBs for handling different types of problems) or multiple CCBs (each dealing with a subsystem of the project). In some cases there can be a super CCB to coordinate the activities of the CCBs and to act as an arbitrator to solve conflicts between CCBs of equal status and authority.

Irrespective of the size and nature of the CCBs, their function is the same: to control and manage change. To manage and control change in a software system, the CCB should be comprised of people who have knowledge about the system (its technical, managerial, and economic aspects) and the effect and consequences of their decisions on the system.

The CCB must be composed of representatives from all affected organizations or departments (stakeholders). It may contain such members as a representative from the SCM group (preferably the CMO); representatives from the project team (project leader or his representative), QA group, company management, marketing department, or project management; members of the functional or user community;

developers; test group, design group, interface group, documentation group, or operation personnel; or database administrators. In some cases, the CCB must include the client's representatives. The members of the CCB should be senior people who can speak for their respective departments. Also, there should be a provision by which the CCB can summon individuals (like the change initiator, the person who conducted the analysis, or outside experts) if their presence is required for better decision making. The ideal size of the CCB is around seven, but it can be more or less depending on the organization. When all project groups are not represented, it is the members' responsibility to ensure that other groups are aware of the CCB's actions. This can be accomplished by the SCM recording minutes of the meeting and circulating it among all the concerned parties.

Although all members of the CCB might not agree to each and every change, it is certain that some change will affect every member of the board at some time. It should not be difficult to recall past experiences when an unwise or costly mistake could have been avoided if the right people had known about a proposed change. The chair of the CCB must be from project management—a person who can unambiguously resolve conflicts within the board and enforce the board's decisions on the project. Decisions on change implementation and CI promotion translate directly to fundamental project cost, schedule, and quality issues.

The CCB will find that its efforts are an infuriating exercise in futility if their decisions are continually reversed or ignored by an outside entity with the real decision-making authority. Do not let this happen; put that entity in charge of the board. By doing so, those who have decision authority are directly coupled to those who have expertise on the details. Decisions of the CCB should be reached by consensus whenever possible. The group dynamics must reflect the cooperative nature of a development project. The chair must nurture this cooperative vision and take unilateral action only when all other methods have been exhausted.

Functions of the CCB

As stated in the IEEE definition, the main function of the CCB is to evaluate and approve or disapprove the CRs and PRs that have been initiated or filed. The CCB will also see to it that the approved changes are implemented in the correct manner. The CRs and problem reports are evaluated before submitting them to the CCB. This evaluation is necessary because it will save a lot of time and effort. Also, there are some tasks that are better accomplished by a single person than a team. So the presubmission evaluation should be done by a qualified professional, who knows the subject well. Assigning the right person to this task is the duty of the CMO. The evaluation report along with the CR or problem report is submitted to the CCB.

The CCB is comprised of members who are quite senior and have other responsibilities and whose time is valuable. Accordingly, speedy resolution of the issue is a must; to attain speedy resolution and better decision making, the facts should be presented to the CCB in a clear and concise manner. It is a good idea to circulate the agenda of the CCB meeting and the issues and the supporting documents to the members so that they can come prepared for the meeting. This is a task that has to be done by the CMO. The CCB members will evaluate the requests for their technical feasibility, economical viability, impact on marketing, and other factors.

Depending on the pros and cons, the committee will decide to approve, reject, or defer the CRs.

During the presubmission evaluation, the analysis focuses on factors such as the impact of the program on other programs, the cost of implementation, the skills required for implementing the change, and the time required. During the CCB meeting, these factors, along with other issues such as how a particular change is going to affect the system release schedule, how the change will affect the marketing strategy, and how it will affect the quality of the system, will be evaluated. In other words, the CCB members will decide on each request looking at the overall picture. The concerns and issues the CCB discusses include the following:

- *Operational impact:* What will be the effect of this change on the final product?
- *Customer approval:* Will the change require customer approval? Is it a major change?
- *Development effort:* What is the impact of the change on interfaces and internal software elements of the final system?
- *Interface impact:* Will the change affect the established interfaces of the system?
- *Time schedule:* At what point is this change incorporated? What is the time for incorporation with minimal impact on cost and schedule?
- *Cost impact:* What is the estimated cost of implementing the change?
- *Resources impact:* What resources—infrastructure, skill, people, etc.— will be required to implement the change?
- *Schedule impact:* How will the processing and incorporation of the change affect the current schedule?
- *Quality impact:* How will the change affect the quality and reliability of the final product?
- *Feasibility:* With all the above factors, can this change be made in an economical manner?
- *Risks involved:* What is the risk of implementing the change? What is the risk of not implementing or deferring the implementation?

For example, a CR, if implemented, will delay the system release, but it is a user interface change that the marketing department feels will improve the sales. As a result, the CCB has to decide whether to delay the release for better sales or to go ahead with the release and incorporate the CR in the next version. Overall, the decisions made in a CCB meeting are strategic in nature, even though a good technical understanding is necessary to make those decisions.

Functioning of the CCB

The CCB should have a chairman. Usually the project management representative is given this post. In some organizations, however, the members are assigned this post on a rotating basis. The CCB should meet at the intervals specified in the SCM plan. There should be a provision to call an emergency meeting if need arises. This is because certain CRs may require immediate action and cannot be delayed until the next scheduled CCB meeting.

The minimum number of people who can make a decision must also be specified. The rules for the functioning of the CCB should be formulated. How will the CCB decide on an issue? Is it by vote, and if so, what will be done in the case of a tie? These things should be specified in the SCM plan. If they are not specified in the SCM plan, then they should be addressed at the first meeting of the CCB.

Another important point is that all transactions that happen during the CCB meetings should be recorded. The minutes of the meeting should be circulated among the CCB members. The format and the style of the meeting minutes can be formal or informal, but they should contain at least the following information:

- Members present;
- Date of the meeting;
- Agenda of the meeting;
- Action taken report (ATR) by the CMO (status of the CRs and other SCM activities since the last CCB);
- CRs (CR number and evaluation document number) discussed at the meeting;
- Discussion details and decisions;
- Distribution list.

The approved changes will be assigned to a qualified person or team for making the changes. If a CR is rejected, then the change initiator will be notified about the decision and the reasons for rejection. The initiator can resubmit the request or file an appeal, if he or she feels that the reasons are not satisfactory. The deferred CRs are filed for later discussion, and the decision is conveyed to the initiator.

In the case of approved changes, the CCB will assign the task of implementing the change to someone or assign the change management process to the CMO. In such a situation, the CMO will assign the task to qualified person(s) and perform the necessary actions to complete the change management process (review, approve, promote, and baseline). In the next meeting of the CCB, the CMO will present the ATR on the CRs that were approved and implemented.

As discussed earlier, the growing popularity of SCM tools, which allow CCB meetings to be conducted electronically, is slowly eroding the need for and importance of physical CCB meetings.

Summary

This chapter discusses configuration control and why it is needed and how it is done. Further, it details the reasons for change and how a change is requested, processed, and implemented. It also describes the benefits of automating the change management process.

In addition, we consider the problem reporting and tracking process and discuss defect prevention. Strictly speaking, defect prevention does not come under the purview of SCM. In the author's opinion, however, because defect prevention is closely related to SCM, the SCM team must perform its associated tasks.

An integral part of any configuration control system is the CCB. Therefore, the chapter explains the composition, functions, and working of the CCB and highlights

that, as change management and SCM tools become more and more popular, a lot of activities that were performed by the SCM team members are now automated. This automation will reduce the monotonous and repetitive nature of change management and help make the configuration control function easier. Moreover, it will improve development productivity, because automation will allow people to concentrate more effort on developmental activities.

References

[1] *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610-1990), IEEE Standards Collection (Software Engineering)*, Piscataway, NJ: IEEE, 1997.

[2] Alain Abran, A., and Moore, J. W. (eds.), *SWEBOK: Guide to the Software Engineering Body of Knowledge (Trial Version)*, Los Alamitos, California: IEEE Computer Society, 2001.

[3] Bays, M. E., *Software Release Methodology*, Upper Saddle River, NJ: Prentice Hall PTR, 1999.

[4] Cagan, M., and D. W. Weber, "Task-Based Software Configuration Management: Support for 'Change Sets' in Continuus/CM," Technical Report, Continuus Software Corporation, 1996.

[5] Burrows, C., S. Dart, and G. W. George, *Ovum Evaluates: Software Configuration Management*, London: Ovum Limited, 1996.

[6] Weber, D. W., "Change Sets Versus Change Packages: Comparing Implementation of Change-Based SCM," *Proc. 7th Software Configuration Management Conf. (SCM7)*, Boston, MA, May 1997, pp. 25–35.

[7] *IEEE Standard for Software Anomalies (IEEE Std-1044–2009)*, Piscataway, NJ: IEEE, 2009.

Selected Bibliography

Ben-Menachem, M., *Software Configuration Management Guidebook*, New York: McGraw-Hill, 1994.

Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, 1992.

Davis, A. M., *201 Principles of Software Development*, New York: McGraw-Hill, 1995.

Gill, T., "Stop-Gap Configuration Management," *Crosstalk: The Journal of Defense Software Engineering*, Vol. 11, No. 2, February 1998, pp. 3–5.

Humphrey, W. S., *Managing the Software Process*, New York: Addison-Wesley, 1989.

IEEE Standard for Software Configuration Management Plans (IEEE Std-828–1998), Piscataway, NJ: IEEE, 1998.

IEEE Standard for Configuration Management in Systems and Software Engineering (IEEE Std-828–2012), Piscataway, NJ: IEEE, 2012.

IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition), Piscataway, NJ: IEEE, 2003.

Intersolv, "Software Configuration Management for Client/Server Development Environments: An Architecture Guide," White Paper, Intersolv, 1998.

ISO, "Quality Management-Guidelines for Configuration Management," Technical Report No. ISO 10007:1995(E), Geneva, International Standards Organization, 1995.

NASA, "NASA Software Configuration Management Guidebook," Technical Report SMAP-GB-A201, NASA, 1995.

Peters, J. F., and W. Pedrycz, *Software Engineering: An Engineering Approach*, New York: John Wiley & Sons, Inc., 2000.

Pfleeger, S. H., *Software Engineering: Theory and Practice (4th Edition)*, Prentice Hall, 2009.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

Weber, D. W., "Change-based SCM Is Where We're Going," Technical Report, Continuus Software Corporation, 1997.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

CHAPTER 8

Status Accounting

Introduction

CSA is an element of CM that consists of the recording and reporting of information needed to manage a software system and its characteristics effectively. This information includes a listing of approved configuration identifications, the status of proposed changes to the configuration, and the implementation status of approved changes [1]. In other words, the status accounting function is the recording and reporting of information needed to manage CIs effectively, including, but not limited to, a record of approved configuration documentation and identification numbers, the status of proposed changes, the implementation status of approved changes, the status of pending or open CRs and PRs, and the build state of all units of the CIs. Status accounting is the record keeping element of SCM, and the status accounting records are the means for SCM to report the state of the software product's development to the project management, company management, and the customer.

The aim of status accounting is to keep managers, users, developers, and other project stakeholders informed about the various configuration stages and their evolution. This implies three basic tasks: data capture, data recording, and report generation. The status accounting activity is important for maintaining the continuity of the project and avoiding duplication of effort.

The effectiveness of CM is closely linked to the flow and availability of configuration information about the product. Information is collected while performing activities associated with the CM processes (planning and management, identification, change management, and verification and audit). CSA correlates stores, maintains, and provides readily available views of this organized collection of information. CSA provides access to accurate, timely information about a product and its documentation throughout the product life cycle. CSA involves the storage and maintenance of the following:

- Information about the configuration documentation (such as document identifiers and effective dates);
- Information about the product's configuration (such as part numbers or changes installed in a given unit);
- Information about the product's operational and maintenance documentation (such as the documents affected by each change and their update status);
- Information about the CM process (such as the status of CRs).

The status accounting activity designs and operates a system for the capture and reporting of necessary information as the life cycle proceeds. As in any information system, the configuration status information to be managed for the evolving configurations must be identified, collected, and maintained. Various pieces of information and measurements are needed to support the SCM process and to meet the configuration status reporting needs of management, software engineering, and other related activities. A good status accounting system should be able to answer questions like the following and many more:

- What is the status of an item?
- Has a particular CR been approved?
- What is its status of pending or open CRs and PRs?
- What items were affected by a particular CR?
- When was the CR approved, and who approved it?
- Who performed the change for a particular CR, and when was it completed? Who reviewed it? Who approved it?
- Which version of an item implements an approved CR?
- What CRs are assigned to whom?
- How many high-priority CRs are currently not implemented?
- What is different about a new version of a system?
- How many CRs are initiated each month, and what is the approval rate?
- How many PRs are filed each month, and what is the status of each of them?
- What are the major causes of the problems or defects?

The status accounting of the CIs can be compared with bank accounts, where each CI is an individual account. All transactions that happen to the CI and all activities that are performed on the CI are recorded. So individual transactions can then be tracked through each account as they occur.

Some form of automated tool support is necessary to accomplish the data collection and reporting tasks. This could be a database capability, such as a relational or object-oriented database management system. This could be a stand-alone tool or a capability of a larger, integrated tool environment.

Status Accounting Information Gathering

Since CSA information is a byproduct of all the other CM processes, the effectiveness of CSA is dependent on the quality of CM implementation, supported by CM processes that ensure the information is systematically recorded, safeguarded, validated, and disseminated. Decisions on the information to be captured in the CSA system should be based on such factors as the nature of the product, the environment in which the product will be operated, the anticipated volume and complexity of change activity, and the information needs of the customer(s) [2].

The procedure of tracking the status of the CIs should be established early enough in the software development process to allow for data gathering. Also, the system should be designed in such a way that the SCM activities will update the

status accounting database rather than the person in charge of the status accounting function collecting the data regarding each and every change that is happening.

If control mechanisms are built into the process that make updating the status accounting database a prerequisite for further processing, the data in the database will be current and complete. For example, when a CR is initiated, that information is recorded in the database and if a check is made to ensure that the details of the CR have been entered in the database before it is forwarded to the CCB, the status accounting function can be performed more effectively.

For this process to happen, people should be aware of what to do and how to update the status accounting database. This is beneficial to the people who update the information because they are the same people who will be asking for information about the status of the various items at a later stage.

Status Accounting Database

The status accounting database is established to receive and process the data collected regarding the evolution of the various CIs of the product during the different phases of the software development life cycle. The amount of data collected and the level of detail will depend on the size, complexity, and nature of the project. The primary data of interest reflect knowledge of dates of start and completion of design and builds. Also important is accurately knowing the changes that are being made or that have been made and incorporated so that the up-to-the-minute status of a configuration item can be known. The following list gives the necessary information for a simple status accounting report:

- CI name and identification number;
- Name of the next higher level CI;
- Design start date;
- Design approval date and revision number;
- Coding start date;
- Coding finish date;
- Testing start date;
- Testing finish date;
- Build start and finish dates and revision number;
- System merge date;
- System delivery date and revision number;
- CR date, CR number, and requestor's name;
- Change disposition date;
- Change incorporation date, implementer's name, and revision number.

CRs also have IDs and descriptions. The database is the primary reference point for anything one may need to know or report about the project. The database thus should capture as much information as possible. If the database is made an integral part of the development environment, the details necessary for effective status accounting can be captured automatically as and when it happens. This will

greatly reduce the workload of the SCM team. The database should be secured and protected from tampering by authorized or unauthorized persons during the input of date, query or generation of reports.

An important feature of the database is the ability to trace the software system upward and downward. This is the capability to track the relationship of the software requirements down through the various levels (e.g., system, program, module, and unit—a process known as drill-down in knowledge management terminology—or in the opposite direction (i.e., track the relationship from the smallest element (unit) upward through module, program, system, design, and ultimately, the requirements). This traceability feature is most helpful during audits and for determining the impact of a change on all the interrelated elements.

Importance of Status Accounting

Status accounting refers to the information management (or data management) functions in the SCM system. For each CI designed, developed, reviewed, approved, released, and distributed, the activities that are done and other information such as how they were done; why, where, and when they were done; and who did them have to be recorded.

These details will be useful for everyone involved in the project in various ways. The information needs of a developer are different from that of a project manager, but each and every member of the project team and the support functions will need at least some of the information. Status accounting is the information gathering and dissemination component of SCM. It is also used by management in decision making to monitor the progress of the project, and it can help identify problems before they become critical so that project management can take corrective actions.

The information provided by the status accounting function helps project management identify problems, pinpoint the source of the problem(s), and take corrective action before the situation gets out of hand. From the reports that are produced and by making ad hoc queries, project management can determine how the project is performing and compare the performance against the plan. One can also look at the types of changes, the rate of changes, the causes of the changes, the cost, and many other factors and take the necessary actions.

Status accounting reports are invaluable during the maintenance phase. To understand and identify the cause of a problem, one needs to know the history of the CI. For example, consider a program that was working until last week but is not working now. The easiest way to find out why is to identify the changes that were made to the program since last week. In situations like this, the information provided by status accounting helps resolve the problem faster.

Similarly, the information provided by the status accounting function is useful in determining the performance characteristics of the project such as number of CRs, approval rate, number of PRs, average time for a change resolution, average implementation time, and cost of implementing a change. This information will help when evaluating the performance of the project and when comparing different

projects. Also, these details will help to fine tune the estimation and costing procedures of the organization.

So when the SCM system is designed, the information that has to be gathered by the status accounting function should be identified and selected, keeping in mind all of the uses mentioned here. A good status accounting system should provide information that is accurate, relevant, and timely.

Status Accounting Reports

As previously discussed, the major functions of status accounting are to record and report information needed to manage a software system and its characteristics effectively. Reported information can be used by various organizational and project elements, including the development team, the maintenance team, project management, and QA activities. Reporting can take the form of ad hoc queries to answer specific questions or the periodic production of predesigned reports. Some information produced by the status accounting activity during the course of the life cycle might become QA records. In addition to reporting the current status of the configuration, the information obtained by status accounting can serve as a basis for various measurements of interest to management, development, and SCM. Examples include the number of CRs per CI and the average time needed to implement a CR [3]. Even though it is not possible to anticipate all possible information requests, there are several reports that every system must have. These include the change log, the progress report, the CI status report, and the transaction log. We will look at each of these reports in some detail a little later.

The factors that should be considered while designing the reporting requirements and reports of a system include the following:

- The audience for the report;
- The information contained in each report;
- The need for a routine report or a report provided on an ad hoc basis;
- The frequency of the report;
- The distribution list.

Examples of routine reports are, as we have seen, the change log and transaction log. Some examples of ad hoc reports are listed as follows:

- A list of all CRs that have been approved but not implemented;
- A list of all CRs initiated in the last four months;
- A list of how many people are working on a particular CR;
- A record of how much time was needed to implement a particular change;
- The number and details of CRs that are pending.

Ad hoc reports are generated when a user requests particular information not included in the routine reports.

Let's take a look at some of the most common routine reports.

Change Log

The change log should contain all information about the CRs in the system. The usual distribution frequency is monthly. This report should contain information such as CR number, status, originator's name, impacted items, origination date, description of change, and implementer's name.

Progress Report

The progress report, which is a summary of development progress since the last report was issued, is used primarily by management to monitor the progress of the project. This report should include information such as the reporting period (from and to dates), the task ID[1], a brief description of the work performed during the period on the task, and the status of the task (e.g., complete or percent completed).

CI Status Report

This report is prepared to summarize the status of all CIs in the system and should include information such as a list of the CIs, description, and location of the CIs (the controlled library where they are stored). The CI description should include the name, version number, and details of dependent items.

Transaction Log

This log contains the transactions that have happened to items, recorded in chronological order. The log should contain details such as transaction number, date, originator (person who is making the entry), nature of the entry (what the entry regards), affected items, activity (e.g., CR, CCB approval, analysis, and PR), description, participants (people who are involved), impacted items (items affected by the activity), and remarks.

The objective of the transaction log is to find out what happened during a specific period, say, "What were the activities done on mm/dd/yy?" Here, the idea is not to provide a detailed description of how things were done, but to give someone a snapshot of what happened during a given period.

Status Accounting and Automation

SCM tools fully automate the status accounting function. We have seen that SCM tools capture all SCM-related information as and when it happens and does so automatically. The SCM tools store this information in a database, so retrieval of the information is fast and efficient. Also, the information can be generated in any format the user wants. SCM tools can be used to generate routine reports.

1. A task is the result of a CR implementation. An approved CR can result in one or more tasks that can be assigned to one or more persons or teams. When the task is created the CMO creates a task ID and associates it with the CR.

However, SCM tools deliver their full potential when they are used for generating ad hoc reports. Users can query the system for any information they require and get answers immediately and in the requested format.

Without SCM tools, SCM teams have to go through data such as that contained in change logs, defect logs, CRs, and transaction logs to correlate and collate bits and pieces of information from different sources to create a report that satisfies the user's requirements. This is a tedious and time-consuming process that increases the workload of the SCM team if users start asking complex queries that involve more than one source.

An SCM tool is an ideal solution to this situation. It is quick, accurate, and customizable. In the case of an SCM tool, all SCM-related information is stored in relational databases, making information retrieval easy and quick. Also, in the case of a tool, the information will be up-to-date, as the events are recorded and the information is captured as the activities occur.

Another advantage of using tools is the variety of information that can be retrieved. For example, one can obtain information on such topics as all pending CRs, all completed CRs, all CRs completed between such-and-such a date, all CRs initiated by a particular developer, and all CRs implemented by a particular person. The beauty of this system is that no additional cost or extra effort is required to produce these different reports. Most SCM tools will have a set of standard reports and then the facility to query the SCM database for other details. Most tools have the facility to display the information in graphical form also.

Thus, SCM tools greatly enhance the capabilities of the status accounting function because they can record and retrieve minute details with speed and accuracy to satisfy ad hoc queries. This is especially true when different users of the SCM system have different needs. The project manager's information requirements are different from that of the developer's. What the change initiator wants to know will be different from what the project leader or a QA person wants to know. With a manual system, people have to wait while the information is compiled from records, whereas with a computerized system, retrieval is merely a matter of running a query.

Another advantage of a software tool is that the people concerned can be given read-only access so that they can query the system and get the answers they need. Here, once again, the author would like to stress the need for a computerized system. Manual systems are fine for routine reports. However, an interactive system can reduce the workload of the SCM staff because the people who want the information (and have the necessary authority) can log on to the system and get the information required. Also, routine reports provide static information. With an interactive system, on the other hand, users can view the reports, and if they require more information, they can drill down and get the details they want. In this way, SCM information is more effectively used, leading to better, well-informed decisions. If an SCM tool does not support flexible and customizable reporting features, it is of limited value.

As previously mentioned, SCM tools completely automate the status accounting process and make status accounting easier and accurate. However, the level of automation possible and the capabilities offered differ among SCM tools. Most organizations use some sort of tool for performing SCM (at least tasks like change management, defect tracking, and build management), and complete manual systems are very rare nowadays. However, to take full advantage of automation, organizations

need to turn to high-end sophisticated SCM tools (which cost a fortune)—and even then, at least some of the SCM functions will require human operation.

Next, we examine some of the common report categories that are supported by most SCM tools.

Change and Problem Tracking Reports

These reports contain details such as who made the change and when, who initiated it, change history, and CR status. Here, the advantage, as mentioned previously, is that the users can tailor the information retrieved in any format that they would like. So one can generate reports of all unassigned CRs; all pending CRs; all CRs assigned to a particular person; and CRs sorted by such factors as date, severity, priority, classification, completion date, and status.

Difference Reporting

It is important to keep track of the differences between versions and releases, because doing so will make it easier to incorporate changes from one version to the next. Most SCM tools have the facility to generate difference reports, which will contain the differences (changes) between two versions of an item or set of items.

Ad Hoc Queries

The usefulness of having ad hoc querying capabilities can never be overstated. Ad hoc queries allow the users of an SCM system to get the information they want, when they want it, and in the form they want. The reporting tools are so advanced that many of them have graphical user interfaces that help users write their own queries by choosing the items in which they are interested. Also, these reporting tools have drill-down features, so that a user can drill down from a summary report to the level of detail required. This feature is particularly useful for project leaders and management.

Journals

The journal feature distinguishes SCM tools from the manual status accounting process. A journal records all events that happen to all configuration items as they occur, thus providing users with a complete and comprehensive picture of what happened during a particular period in time. This is a much advanced and more comprehensive version of manual transaction logs.

Journals provide audit trails that can be used for a variety of purposes including configuration audits. The advantage of having the journal details in a database is that the information contained in it can be manipulated at will. Accordingly, one can recreate all of the events that happened during the transition of a configuration item from, say, version 1 to 6 or the details of activities performed by a certain developer. The advantage here is that the information can be retrieved quickly and without any extra effort.

Summary

Status accounting is a recording activity that serves as a follow-up to the results of the SCM activities of configuration identification and change control. It keeps track of the current configuration identification documents, the current configuration of the delivered software, the status of the changes being reviewed, and the status of the implementation of approved changes.

The status accounting function plays a vital role in the efficient management and control of projects by providing the necessary information to project management and the project team. SCM tools automate the status accounting function and help provide users with information that is accurate, timely, and relevant.

References

[1] *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610–1990), IEEE Standards Collection (Software Engineering)*, Piscataway, NJ: IEEE, 1997.

[2] EIA, *National Consensus Standard for Configuration Management (EIA-649)*, Arlington, VA: Electronics Industries Alliance, 1998.

[3] Alain Abran, A., and Moore, J. W. (eds.), *SWEBOK: Guide to the Software Engineering Body of Knowledge (Trial Version)*, Los Alamitos, California: IEEE Computer Society, 2001.

Selected Bibliography

Babich, W. A., *Software Configuration Management: Coordination for Team Productivity*, Boston, MA: Addison-Wesley, 1986.

Ben-Menachem, M., *Software Configuration Guidebook*, London: McGraw-Hill International, 1994.

Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, 1992.

Bersoff, E. H., V. D. Henderson, and S. G. Siegel, *Software Configuration Management, An Investment in Product Integrity*, Englewood Cliffs, NJ: Prentice-Hall, 1980.

EIA, *National Consensus Standard for Configuration Management (EIA-649)*, Arlington, VA: Electronics Industries Alliance, 1998.

IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition), Piscataway, NJ: IEEE, 2003.

Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

"Software Configuration Management: A Primer for Development Teams and Managers," White Paper, Intersolv, 1997.

"Software Configuration Management for Client/Server Development Environments: An Architecture Guide," White Paper, Intersolv, 1998.

CHAPTER 9

Configuration Verification and Audits

Introduction

The objective of configuration verification and audits is to verify that the software system matches the CI description in the specifications and documents and that the package being reviewed is complete. According to EIA-649 [1], configuration verification and audit establish that the performance and functional requirements defined in the configuration documentation have been achieved by the design and that the design has been accurately documented in the configuration documentation. The purpose and benefits of the process include the following:

- Ensuring that the product design provides the agreed-to performance capabilities;
- Validating the integrity of the configuration documentation;
- Verifying the consistency between a product and its configuration documentation;
- Determining that an adequate process is in place to provide continuing control of the configuration;
- Providing confidence in establishing a product baseline;
- Ensuring a known configuration as the basis for such things as operation and maintenance instructions, training, and spare and repair parts.

Once the software has been designed, developed, and tested, it is necessary to establish that the software product has been built in accordance with the requirements and that the software is correctly represented in the documentation that is shipped along with the software. A configuration audit (CA) is a check to verify that the product package contains all of the components it is supposed to contain and performs as promised.

Configuration verification and audits enable the developer and the customer to agree that what has been designed has been built and that the testing applied to the software product CIs proved that the requirements of the software requirements specification (SRS) for each CI were met. The CAs are performed after software integration and testing. In some cases, the software will be audited with the hardware—after system integration. Reviews are an iterative activity that start with receipt of a contract and culmination of an agreement and end with delivery of the software product and its associated documentation [2].

The configuration verification and audit process includes the following:

- Configuration verification of the initial configuration of a CI, and the incorporation of approved changes, to assure that the CI meets its required performance and documented configuration requirements;
- CA of configuration verification records and physical product to validate that a development program has achieved its performance requirements and configuration documentation or the system/CI being audited is consistent with the product meeting the requirements.

The common objective is to establish a high level of confidence in the configuration documentation used as the basis for configuration control and support of the product throughout its life cycle. Configuration verification should be an embedded function of the contractor's process for creating and modifying the CI or CSCI. Validation of this process by the government may be employed in lieu of physical inspection where appropriate [3]. Inputs to the configuration verification and audit activity are listed as follows:

- Configuration, status, and schedule information from status accounting;
- Approved configuration documentation (which is a product of the configuration identification process);
- Results of testing and verification;
- Physical hardware CI or software CSCI and its representation;
- Build instructions and tools used to develop, produce, test, and verify the product.

Successful completion of verification and audit activities results in a verified system and CI(s) and a documentation set that may be confidently considered a *product baseline*. It also results in a validated process to maintain the consistency of product to documentation.

Many organizations do not perform CAs [functional CAs (FCAs) and physical CAs (PCAs)] and instead perform audits such as market readiness reviews (MRRs), test readiness reviews (TRRs), or alpha and beta testing.

MRRs are conducted to confirm that the distribution, service, maintenance, technical support, and field people are ready; that the installation, operation, and troubleshooting manuals are ready; and that product tests and trial runs were successful, among other purposes.

TRRs are conducted to evaluate preliminary test results for one or more CIs, to verify that the test procedures for each CI are complete, to comply with test plans and descriptions, to satisfy test requirements, and to verify that a project is prepared to proceed to formal testing of the CIs. TRRs will be held for each application of a release at the completion of the software integration test and at the completion of the functional validation test. There are three levels of TRRs at the application level, defined as follows:

- *Development TRR:* Informal TRR conducted following successful completion of unit or module testing of a given application;
- *Project TRR:* Formal TRR conducted following successful completion of the software integration test (SIT) of a given application;

- *Enterprise TRR:* Formal TRR conducted following successful completion of the functional validation test (FVT) of a given application.

Alpha testing is done when the system or product has a lot of new previously untested features. Since there is a lot of untested functionality, the development team might be uncomfortable proceeding with the final testing and release of the product until they get a feedback from a limited number of users or customers. So developers use alpha testing primarily to evaluate the success or failure (or acceptance) of the new features incorporated into the system.

Beta testing is required when the development team decides that some level of customer evaluation is needed prior to the final release of the product. In the case of the beta testing, the developers are no longer looking for user inputs on functionality or features. The product has all the functionality incorporated in it, so the development team will be looking for the beta testers to uncover bugs and faults in the system. Unlike alpha testing, beta testing is done on a much larger scale (i.e., the number of people doing the beta testing will be much higher than that for alpha testing). Usually, companies distribute the beta releases free of cost to the people who have enrolled for the beta testing program, and in many cases, the beta versions will be available for download from a company's Web site. New products will have alpha testing followed by beta testing. In the case of new versions of existing products, however, either alpha or beta testing is done.

It is quite clear that these tests and reviews are not as comprehensive and thorough as CAs and that they do not provide the same kind of assurance that products are built according to specifications and complete in all respects. CAs, on the other hand, provide objective evidence of products' and processes' compliance with standards, guidelines, specifications, and procedures.

Software Reviews

A review is a process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval [4]. Technical reviews are a series of system engineering activities by which the technical progress on a project is assessed relative to its technical or contractual requirements. The reviews are conducted at logical transition points in the development effort to identify and correct problems resulting from the work completed thus far before problems can disrupt or delay the technical progress. Reviews, as we have seen, are performed many times during the development process at least, at the completion of each phase and sometimes more often. Reviews—which include system requirements reviews, software requirements reviews, design reviews (preliminary design reviews and critical design reviews), and code reviews—provide a method for the performing activity and tasking activity to determine that the development of a CI and its documentation has a high probability of meeting contract requirements.

On the other hand, audits are performed at the completion of the product to make sure that the product is complete in all respects and that the development has been performed in conformance with the development standards and guidelines

Configuration Verification

Configuration verification is an ongoing process that is common to CM, systems engineering, design engineering, manufacturing, and QA. It is the means by which the design solution is verified. Verification that a design achieves its goals is accomplished by a systematic comparison of requirements with the results of tests, analyses, or inspections. The documentation of a product definition must be complete and accurate enough to permit reproduction of the product without further design effort. The design of a product must be verified to ascertain that it has achieved specified requirements and desired goals, that the documentation of the design is accurate, and that the product can be produced from the documentation [1].

Conceptually, verification occurs in sequence by first determining the acceptability of the design and then confirming that the documentation portrays that design. In practice, it may be accomplished in separate events or audits. It is often more practicable to verify these aspects incrementally during the course of the definition phase and to incorporate the verification into the design and development process flow, so that it occurs on a continuous basis.

Verification methods should be carefully planned to ensure that all requirements are addressed and that the individual verification methods chosen are appropriate. Requirements analysis and test tools that flow down, account for, and verify all attributes facilitate the design verification process. Results are typically recorded in a matrix indicating each discrete requirement, the method of verification, the verification procedure, and the verification results. The design output, consisting of the complete set of design information, must be accurately documented to permit reproduction of the product without further design effort. Beyond this fundamental requirement, other factors (such as the need to procure from other sources or future maintenance needs) may influence the content and formality of documentation. A product should be able to be produced from its documentation with confidence that it will meet all requirements.

A software product should also be in compliance with published design and coding standards so that it can be maintained, modified, and upgraded. In addition, the following should be verified:

- The documentation library control system;
- The uniqueness of the product identifier;
- The validity of interfaces;
- The internal audit records of CM processes and procedures.

Verifying the documentation determines that it is adequate for its intended purposes and accurately reflects compliant design. The verification of design and documentation must be planned to permit its accomplishment at minimum cost. In complex physical products, the comparison of the documentation with the prototype or test article can often be accomplished incrementally, during assembly of the article, to avoid the need for later disassembly. These verifications are considered complete upon resolution of discrepancies or departures found and correction of associated documentation.

The When, What, and Who of Auditing

Audits are the means by which an organization can ensure that software development has been performed in the correct way—that is, in conformance with development standards and guidelines. The software configuration auditing activity determines the extent to which an item satisfies the required functional and physical characteristics. Audits vary in formality, but all audits perform the same function—they check the completeness of the software system or product. Any anomalies found during an audit should not only be corrected, but the root cause of the problem should be identified and corrected to ensure that the problem does not occur again. (Here, the defect prevention methods described in Chapter 8 are quite useful.)

Before the release of a product baseline, an FCA and a PCA of the CIs are usually conducted. The FCA ensures that the functions defined in the specifications are all implemented in the correct manner. The PCA determines whether all the items identified as being part of the CI are present in the product baseline.

A software audit is an activity performed to evaluate independently the conformance of software products and processes to applicable regulations, standards, guidelines, plans, and procedures [4]. An audit is usually done at the end of a phase in the development life cycle. Before the development proceeds to the next phase, it is a good practice to conduct an audit so that the development team has the satisfaction of knowing that it is working on something that is complete and approved. The reality, however, is that audits are usually performed only before a system release. This is because the system release is the one that will go to the customer.

Conducting the audit prior to final release gives the company and the customer the satisfaction of knowing that what they are delivering or getting is complete in all respects and meets the requirements specified. The CA can be performed before the final release for projects done in-house and when the organization is following all the development standards and guidelines and other QA procedures are in place. Even then, however, every final major baseline or release must be audited. Items supplied by subcontractors must be subjected to a formal auditing process.

Who should perform the CAs? Audits are usually performed by a representative or team of representatives from management, the QA department, or the customer or client. In some cases, the auditing is done by an external agency. An external auditor is best, because the auditing activity requires a very high degree of objectivity and professionalism. The person who conducts the audit should be knowledgeable about SCM activities and functions and technically competent to understand the functionality of the project. The SCM plan should describe the types of audits and reviews to be applied to a specified software project. Where hardware components are also involved, it is necessary to pay special attention to what documents and data will be reviewed and audited and who will represent the hardware and software engineering functions assigned to the project.

Audits may be conducted by the organization responsible for the product development, by the customer, or by a third party designated by the customer. A chairperson representing each party to the audit participates in audit planning and preparation. Audit plans and agendas are reviewed and agreed to prior to the audit. Audits of complex products may be accomplished in a series of incremental audits.

During an audit, audit participants record significant questions, discrepancies or anomalies, and recommended courses of action. Chairpersons review audit findings and determine appropriate actions. Affected parties agree to action items and the plan for effecting their successful closure. Audit minutes provide a record of the audit findings, conclusions, recommendations, and action items. Follow-up occurs until all required action items are complete. The necessary resources and material to perform an audit include the following items to the extent appropriate for the type and scope of audit:

- An audit plan and agenda;
- Adequate facilities and unencumbered access;
- Assignment and availability of personnel;
- Applicable specifications, drawings, manuals, schedules, and design data, test results, inspection reports, process sheets, data sheets, safety procedures, and other documentation as deemed necessary;
- Tools and inspection equipment necessary for evaluation and verification;
- Access to the product(s) and detailed parts to be reviewed.

FCA

The objective of the FCA is to verify that a CI's actual performance agrees with the requirements specified in the requirements definition and system design documents. IEEE [5] defines FCA as an audit conducted to verify that the development of a CI has been completed satisfactorily, that the item has achieved the performance and functional characteristics specified in the functional or allocated configuration identification, and that its operational and support documents are complete and satisfactory. An FCA will prove, right or wrong, that the software test reports correctly state that a given requirement has been met.

The FCA team reviews the test plans, the test data, and the testing methodology to verify that all functional parameters were tested and that the test results were satisfactory. The audit team may ask for additional tests to be conducted, if necessary. Functional audits normally involve a structured and well-defined sequence of tests designed to ensure that the performance of the new or modified item conforms to the requirements in the specification.

The form of the FCA will vary according to the type and extent of change involved. In most cases, the FCA represents a review of the qualification of the item, to ensure that it not only meets the specification requirement but that there are no unintended consequences associated with the change. This process may include some or all of the following forms of test, analysis, or demonstration: environmental tests, to ensure that the new design is suitable for operation within the extremes of the operational requirements; reliability tests; user trials; interfaces with other systems; software testing; and stress testing.

As mentioned, the audit team consists of representatives from management, QA, external experts, and client representatives. An audit team can be an ongoing part of the organization or it can be constituted on an as-needed basis. Sometimes audit teams are from an external agency that specializes in conducting audits. The

composition and structure of the audit team depends on the company and the auditing standards that the company is following.

PCA

The objective of the PCA is to verify that a CI, as built, conforms to the technical documentation that defines it. The PCA is usually done after successful completion of the FCA. A PCA will demonstrate that the documentation for each CI and the software system that will be delivered with the software product correctly describes the functional and physical characteristics of the product and that the software product specification and version description documents are consistent with the software product. The audit team examines the design documentation with the source code and user documentation and any other items that will accompany the final software system. When a PCA is completed, the product baseline is established. In other words, the successful completion of the FCA and PCA are prerequisites to the establishment of the product baseline.

Auditing the SCM System

SCM system audits are carried out to ensure that the implementation of SCM remains consistent with established policy and procedures. System audits are essential to ensure that defined processes are being properly applied and controlled. General aspects that may be considered part of a system audit are listed as follows:

- The operational change control processes, including the CCB function;
- The implementation of change requests;
- The traceability of approved changes to the original specification and requirement;
- The availability of design data and documentation in support of approved changes;
- The traceability of design decisions to the initiating requirement.

The auditing of the SCM system is done by management's representatives, QA personnel, or SCM experts. It is better to have the SCM system audits done by the people who reviewed the SCM plan, because they are familiar with the SCM system that is being practiced and thus are able to do a better job. The SCM system will be audited against the SCM plan and the standards mentioned in the SCM plan. The purpose of auditing the SCM system is to ensure that the SCM system and SCM functions and procedures are being practiced as specified in the SCM plan and to find out areas in the functioning of the SCM system that need improvement.

Role of the SCM Team in CAs

It is the responsibility of the SCM team to schedule the audits and find qualified personnel to perform them. The SCM team also liaises between the audit team and

the development and testing team and ensures that the audit team gets full support from the development and testing team to carry out the audits successfully.

Sometimes, auditors need to question the development or testing team members as part of the audit. The SCM team should act as a facilitator of such meetings and record the points discussed; these minutes should form part of the audit report. In addition, the SCM team should arrange for the infrastructure facilities such as room(s), furniture, machine access, documents, and any other items required by the audit team.

After the FCA and PCA have been completed, the SCM team reviews the auditor's comments and nonconformance reports (NCRs) and initiates necessary corrective actions.

CAs and SCM Tools

SCM tools make the auditing process a lot easier than before. As we have seen, SCM tools capture all SCM-related information in a very comprehensive manner as the activities occur. In addition, the journal reports created by SCM tools record all events that have happened to the CIs, thus creating an audit trail, which can be used by auditors to perform their job. Also, the querying facility of the tools helps the auditing team get any other information it needs.

The automated information gathering abilities of the SCM tools make auditing into an incredibly simple process, because any necessary information can be generated for verification purposes and auditors can confirm whether the system or product they are auditing is complete and meets all the requirements.

Summary

CAs are carried out to ensure that software systems are functioning correctly and to ensure that the configurations have been tested to demonstrate that they meet their functional requirements and contains all deliverable entities.

The two types of audits are PCAs and FCAs. Whereas FCAs authenticate that the software performs in accordance with the requirements and as stated in the documentation, PCAs authenticate that the components to be delivered actually exist and that they are complete in all respects and contain all of the required items.

SCM systems should also be subjected to auditing to ensure that the implementation of SCM remains consistent with established policies and procedures. SCM tools automate most of the auditing tasks and make auditing an easy and painless job.

References

[1] EIA, *National Consensus Standard for Configuration Management (EIA-649-B)*, Arlington, VA: Electronics Industries Alliance, 2011.

[2] Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

[3] U.S. Department of Defense, *Military Handbook: Configuration Management Guidance (MIL-HDBK-61A(SE)-2001)*, U.S. Department of Defense, 2001.

[4] *IEEE Standard for Software Reviews (IEEE Std-1028-2012)*, Piscataway, NJ: IEEE, 2012.

[5] *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610–1990), IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition)*, Piscataway, NJ: IEEE, 2003.

Selected Bibliography

Arthur, J. D., et al., "Evaluating the Effectiveness of Independent Verification and Validation," *IEEE Computer*, Vol. 32, No. 10, October 1999, pp. 79–83.

Ben-Menachem, M., *Software Configuration Management Guidebook*, New York: McGraw-Hill, 1994.

Bourque, P., and R. E. Fairley (eds.), *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

EIA, *National Consensus Standard for Configuration Management (EIA-649-B)*, Arlington, VA: Electronics Industries Alliance, 2011.

IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition), Piscataway, NJ: IEEE, 2003.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

U.S. Department of Defense, *Military Handbook: Configuration Management Guidance (MIL-HDBK-61A (SE)-2001)*, U.S. Department of Defense, 2001.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

CHAPTER 10

SCM: Advanced Concepts

Introduction

This chapter discusses some of the advanced concepts in SCM, building on the basic concepts introduced in Chapter 5 and dealing with some important SCM functions such as interface control and subcontractor control. These topics are particularly important when writing the SCM plan, which is discussed in Chapter 13. This chapter also deals with the concept of software libraries and how important they are for performing SCM activities.

The foundation of SCM is based on three components (introduced in Chapter 5): version control, system building, and release management.

Version Control

We have seen that a version is an initial release or rerelease of a CI. It is an instance of the system that differs in some way from the other instances. A version is usually accompanied by a version description document (VDD). A VDD is a document that accompanies and identifies a given version of a system or component. Typical contents include an inventory of a system or its component parts, identification of changes incorporated into this version, and installation and operating information unique to the version described [1]. VDD is not normally required for each build or internal release of software or a CI. Typically, a VDD is produced only for system-level testing and for the major activities that follow the system level test (e.g., release to client, customer, or marketing, archival purposes, rerelease, etc.). The VDD, which may exist in hardcopy or electronic form, describes new capabilities, known problems, and platform requirements necessary for proper product operation.

Version control is simply the automated act of tracking the changes of a particular file over time. This is typically accomplished by maintaining one copy of the file in a repository, then tracking the changes to that file. The concepts of check-in and check-out (described in Chapter 5) make this possible. Version control has a number of benefits including the following:

- Preventing unauthorized access and modification to files;
- Tracing the evolution of a file from inception to the current state;
- Rolling back to a previous version of a given file in case of a problem or for debugging;

- Comparing two versions of a file and highlighting differences to see what changes are made;
- Providing a mechanism of locking or forcing serialized change to any given file;
- Creating branches that allow for parallel concurrent development and the ability to combine the changes made by different people to a single file at a later stage—merging;
- Maintaining an instant audit trail on each and every file: versions, modified date, modifier, and any additional amount of metadata a system provides and one chooses to implement.

Thus, version control makes it possible to trace the history of all CIs in a system and to recreate any previous version of a file. This capability gives the software organization tremendous power to identify and pinpoint the files that are creating problems while debugging during the maintenance and support phases. When things are not working, the files can be rolled back to previous versions, and the changes made after the successfully working version can be inspected to find out what is causing the error or malfunction.

System Building

IEEE defines a build as an operational version of a system or component that incorporates a specified subset of the capabilities that the final product will provide [1]. The building activity combines the correct versions of software items, using the appropriate configuration data, into an executable program for delivery to customers or other recipients, such as developers, testers, and QA personnel. Build instructions ensure that the proper build steps are taken and in the correct sequence. In addition to building software for new releases, it is usually also necessary for SCM to have the capability to reproduce previous releases for recovery, testing, or additional release purposes.

During the development of a software product the build process will be performed several—or actually, many—times. The developers, testers, and QA personnel will perform builds during the course of the development to see whether the system that is under development performs as expected. When system development is complete and all the items that are required for the final product are debugged, tested, reviewed, verified, validated, and audited, the system build is performed. The system build produces the product that is given to the client or shipped to the customers.

Software is built using particular versions of supporting tools, such as compilers. The two most essential characteristics that are needed for any build process are repeatable and reproducible. In other words, you should be able to recreate the exact product that was created using a build process at a later date. For this to happen, the supporting tools and associated build scripts need to be under SCM control to ensure availability of the correct versions of the tools. The build process and products are often subject to SQA verification. Outputs of the build process might be needed for future reference and may become QA records.

Tools are very useful for selecting the correct versions of software items for a given target environment and for automating the process of building the software from the selected versions and appropriate configuration data. For large projects with parallel development or distributed development environments, this tool capability is a must. Most software development environments provide this capability. These tools vary in complexity and features; some use scripting languages while others employ GUI-oriented approaches that hide much of the complexity of the build facility.

Release Management

The term "release" is used in this context to refer to the distribution of a software CI outside the development activity. The primary purpose of a release is to make the application available to its end users. Thus, a release can be for internal users as well as customers.

Release management is closely tied to build management in that a specific release is essentially a production build of an application. In addition to putting the runtime software in its final form, release management includes the deployment process as well as the update of related metadata that goes into tracking a given version of a software application. When different versions of a software item are available for delivery, such as variants for different platforms or versions with varying capabilities, it is frequently necessary to recreate specific versions and package the correct materials for delivery of the version. The software library is a key element in accomplishing release and delivery tasks. Software libraries are detailed later in this chapter.

Software release management encompasses the identification, packaging, and delivery of the elements of a product such as the executables, documentation, release notes, and configuration data. Software products will be subjected to changes and enhancements on a continuous basis. Hence, one of the main considerations for release management is determining when to issue a release. The severity of the problems addressed by the release and measurements of the fault densities of prior releases affect this decision [2]. Pressure from competitors, entry of new products and technology, and other environmental factors also affect the decision on when to release.

The packaging task must identify which product items are to be delivered and select the correct variants of those items, given the intended application of the product. The set of information documenting the physical contents of a release is known as a VDD, described earlier in the chapter. The package to be released also contains loading or upgrading instructions. The latter can be complicated by the fact that some current users might have versions that are several releases old. SCM tools are needed for supporting these release management functions.

Software released to customers must be comprised of items that have been approved as fit for their intended use. Usually, this requires that the items be fully approved by the CCB, although beta test or prototype releases may be less than fully approved. It also requires that the correct variant of the system be issued to clients. Variants differ only in the platform or language supported and have the

same functionality. So clients who want a Windows version of a product will not be happy if they receive a UNIX version.

Interface Control

In today's environment, interface design has become an important segment of the software engineering process. One is not only faced with the normal computer system-to-computer system interfaces, but other functions such as LANs to WANs or workstations to files servers to mainframes. Those interfaces that affect the software are identified and documented by the systems analyst and, in turn, are placed under configuration control.

Interface describes the functional and physical characteristics required for a common boundary to exist between two or more software products and computer systems that are provided by different organizations or sources. Interface control is the process of identifying, documenting, and controlling all performance, functional, and physical attributes relevant to the interfacing of two or more products provided by one or more organizations. Interface documentation consists of interface control drawings or other documentation that depicts physical, functional, and test interfaces of related or cofunctioning products [3]. An interface control document defines the interfaces that may affect the operation of cofunctioning CIs and is used for control as well as delineating the interface criteria and technical detail necessary to effect an economical and viable interface [4].

For product interfaces external to the enterprise, the SCM system must establish an interface agreement and a mutually agreed documentation of common attributes. Product attributes include defined interfaces with products that are developed, produced, and supplied by organizations outside the enterprise. External interfaces are documented in a product's configuration documentation. To document and control the interface, there must be a relationship between the interfacing organizations.

If the relationship is a buyer-seller relationship, the interface definition is included as part of the purchase agreement (e.g., by reference to a defined catalog item or by use of a control drawing). If there is no direct relationship, an interface agreement is established between the developing enterprises. It delineates procedures for defining and maintaining the common interface. The procedures (for defining complex interfaces and coordinating proposed changes to them) may employ a joint interface control working group. A mutually agreed upon interface definition (including performance, functional, and physical attributes) is typically detailed in an interface document or drawing.

NASA-DID-M200 provides for an interface control plan (ICP) and states in simple terms that the purpose of the plan is to define the process by which the developer defines and manages all external interfaces between the software and all users—both human and software. It may be appropriate to roll out this plan when there are major coordination concerns and risks between the developer and the organizations responsible for the interfacing units [5].

MIL-STD-483B states that interface control is the coordinated activity required to ensure that the functional and physical characteristics of systems and equipment are compatible [6]. The interface control activity is responsible for ensuring that

the configuration identification conforms to the functional interfaces established by system engineering and that the affected CIs are logically compatible and can be operated and supported as needed. The interface activity is also responsible for controlling documentation, including an assessment of the impact of changes to control documentation or changes emanating from other document changes that could affect the interfaces.

Most software specifications and documents define or explain the interfaces between the CI being identified and another CI or computer system. All of these interfaces must be mapped so that everyone on the project will understand what has been defined and will be able to carry out their specified tasks. SCM treats the interface design documents and drawings in the same manner as other documentation, except SCM also provides for assessment of impacts to interfacing entities.

The days of one organization developing all the components of a system are long gone. This is the era of distributed development. Development teams from around the world and from different organizations work jointly to produce a software system. In such cases, interface control working groups (ICWGs) comprised of the representatives of the participant companies or teams are established. This may be necessary to ensure compatibility of all interfacing entities and to establish better communication among the large number of developers and organizations that may be participating in the design effort.

In most cases, an ICWG is formed at the start of the project. It is composed of the interfacing developers and users and the prime developers' SCM activity. The SCM activity provides for the identification of all the interface specifications and documents authorized by the ICWG and when released by the ICWG, places them under SCM control. The change process for an interface document is the same as that of any other CI; the only exception is that the CCB is replaced by the ICWG [7]. The SCM activity will also maintain the status accounting of the documentation and changes and provide periodic reports to the various participating organizations represented on the ICWG. In addition, status accounting will provide the mechanism for requirements traceability to enable the communication of the impact created by such changes as they occur. Tools are now in use that map the entire software system, including interfaces, and delineate the changes that have occurred by some form of reference marking such as version number or version letter. Such information can be acquired online (if SCM tools that capture this information are used) by the SCM activity and the project for immediate information.

Subcontractor Control

A subcontractor is any supplier, distributor, vendor, or firm that furnishes supplies or services to or for a prime contractor. Subcontractor control is another SCM activity that is provided for in the software project's SCM plan. It is most important that the developing organization select qualified developers who, in turn, can demonstrate an adequate understanding of performing the SCM process and can meet the requirements that have been flowed down to them by the prime developer.

CM requirements appropriate to the product being acquired are passed down (flowed down) to subcontractor(s), typically via purchase orders or other subcontract

agreement instruments. Tailoring of requirements for subcontractors is a major SCM planning activity. The performing activity takes on the role of customer (buyer) to the supplier. Suppliers are monitored via data reviews, configuration change management, design reviews, product test results, CAs, and SCM surveillance reviews, as appropriate.

Data reviews typically include assessment of supplier plans, procedures, and configuration documentation. Configuration change management typically includes review of proposed changes to buyer-approved or -imposed configuration documentation. Design reviews assess the supplier's progress and provide a level of confidence that the product, when developed, will meet its specified attributes. Product test results are positive or negative indicators that required attributes will, or will not, be satisfied. CAs verify that the required attributes have been achieved and that the design of the product has been accurately documented. SCM surveillance reviews verify the continuing application of supplier SCM processes.

There can be several categories of development subcontracts, including ones under which (1) full authority is given to design, develop, build, test, and deliver a specified CI or multiple CIs; (2) limited design authority is given, such as when modifying existing software or performing coding and unit testing only; and (3) no design responsibility is given and the software to be delivered is termed as a nondevelopmental item (NDI) or COTS software.

The level of configuration change management exercised by the buyer (prime contractor) ranges from none to total depending on the nature of the product and the conditions of purchase. For a COTS product or NDI, the buyer generally has no control over the product attributes, but can choose not to buy the product.

In the case of a product purchased using a buyer-prepared control drawing, the buyer typically exercises CCA over the specified form, fit, and function attributes. For a product developed to the buyer's specifications, the buyer normally exercises CCA over the product's requirement attributes. The buyer may also exercise control over the product design if more rigorous SCM has been flowed down to the supplier.

Software Library

The software library is the heart of the SCM. It contains everything that is important to a software project: source code, user and system documentation, test data, support software, specifications, project plans, and derived items [8]. The software library must be secure. It must only be accessed in ways that are consistent with sound SCM. Both read access and write access must be controlled, the former to prevent unauthorized disclosure and the latter to prevent unauthorized or accidental change or deletion.

The software library is an important asset to the performance of the SCM process, especially in carrying out change control, release management, and status accounting activities. A software library is defined as a controlled collection of software and related documentation designed to aid in software development, use, or maintenance [1]. Types include development libraries, controlled libraries, and master libraries.

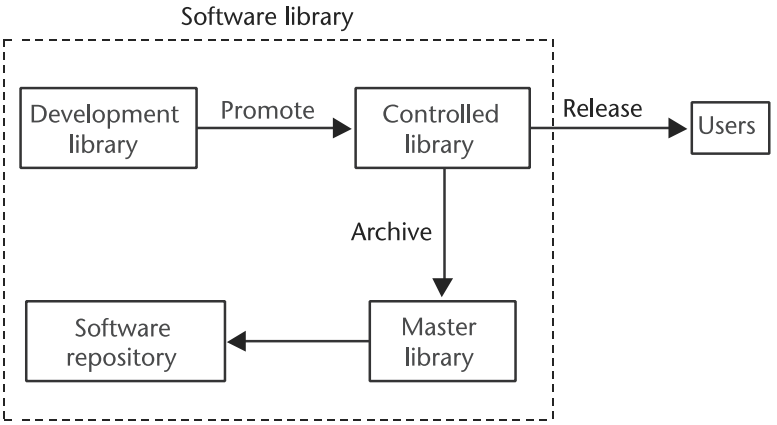


Figure 10.1 Working of the software library.

The development library (sometimes called the dynamic library) holds newly created or modified software entities, data units, or documentation. The production library is the working library for the production of the source code and is usually managed by the developers. This can be a collection of many independent libraries—each owned by different developers. The items in the dynamic library are not under configuration control.

The controlled library (sometimes called the production library) is used for managing current baselines and for controlling changes made to them. It maintains CI units promoted for integration. The controlled library is the entity for retention of approved or released CIs as well as the retention of approved and released software documentation that will be delivered to the customer or distributed to the marketplace.

The master library, also known as the static library, maintains archives of various baselines released for general use. This library contains master copies and authorized copies of software and documentation released for operational use. The items in the master library should not be changed under any circumstances. The master library usually consists of many different physical repositories or storage media. The software repository is the entity that archives software and related documentation at the close of the project. All released documentation and software in the master library should be backed up. The working interfaces of the various software libraries are shown in Figure 10.1.

The status accounting activity will account for all of the software documentation and code in the various library segments and report their status at any given time, including the changes under way, approved, or incorporated.

Summary

The principles of SCM revolve around three key components: version control, system building, and release management. Version control is simply the automated act of tracking the changes of a particular file over time. Version control gives the ability

to trace the history of all CIs in a system and to recreate any previous version of a file. The system build produces the product that is given to clients or shipped to customers. Software release management encompasses the identification, packaging, and delivery of the elements of a product like the executables, documentation, release notes, and configuration data.

Two other important aspects of SCM are interface control and subcontractor control. Interface control is the process of identifying, documenting, and controlling all performance, functional, and physical attributes relevant to the interfacing of two or more products provided by one or more organizations. Subcontractor control is an activity that establishes the procedures for ensuring the quality and completeness of products and components of the system that are being developed by a subcontractor or bought directly from the market. Both interface control and subcontractor control procedures should be described in the SCM plan.

This chapter also describes the different types of software libraries and the role they play in the practice of SCM.

References

[1] *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610–1990), IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition)*, Piscataway, NJ: IEEE, 2003.

[2] Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

[3] EIA, *National Consensus Standard for Configuration Management (EIA 649)*, Arlington, VA: Electronic Industries Alliance, 1998.

[4] U.S. Department of Defense, *Configuration Management Data Interface (MIL-STD-2549)*, 1997.

[5] NASA, *NASA Software Documentation Standard (NASA-STD-2100-91)*, Washington, D.C.: National Aeronautics and Space Administration, 1991.

[6] Department of Defense, *Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs (MIL-STD-483B)*, 1985.

[7] Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, Inc., 1992.

[8] Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

Selected Bibliography

Ben-Menachem, M., *Software Configuration Management Guidebook*, New York: McGraw-Hill, 1994.

Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, Inc., 1992.

Bourque, P., and R. E. Fairley (eds.), *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

Department of Defense, *Military Handbook: Configuration Management Guidance [MIL-HDBK-61A (SE)]*, 2001.

EIA, *National Consensus Standard for Configuration Management (EIA 649)*, Arlington, VA: Electronic Industries Alliance, 1998.

IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition), Piscataway, NJ: IEEE, 2003.

Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

CHAPTER 11

SCM Standards

Introduction

Configuration management got its start in the U.S. defense industry as a technique to resolve problems of poor quality, wrong parts ordered, and parts not fitting, which were leading to inordinate cost overruns. In 1962, the U.S. Air Force published the first standard on CM—AFSCM 375-1. The AFSCM 375-1 identified CM as the key element in the design, development, testing, and operation of the item to be delivered, because CM procedures facilitated better communication and prevented uncontrolled change. In 1964, the National Aeronautics and Space Administration (NASA) developed a CM standard (NPC 500-1) that was based on the AFSCM 375-1, for the design and development of the Saturn V spacecraft. This standard played an instrumental role in the Saturn V and Apollo space programs. During the same time the U.S. Army came out with its version of a CM standard (AMCR 11-26), and in 1965 the U.S. Navy followed suit with its standard NAVMATINST 4130.1 (Configuration Management Policy and Guidance Manual). In 1968, four major standards related to CM were published:

- Department of Defense Directive (DOD D) 5010.19–Configuration Management;
- MIL-STD-480–Configuration Control Engineering Changes, Deviations and Waivers;
- MIL-STD-482–Configuration Status Accounting Data Elements & Related Features;
- MIL-STD-490–Specification Practices.

These standards gave a new thrust to the practice of CM and were integrated into defense contracts, so that not only the military used these standards internally, but also the defense industry (the government contractors and commercial corporations who supplied materials and equipment to the military) started subscribing and implementing these standards.

In 1971, the U.S. Air Force issued MIL-STD-483—Configuration Management Practices for Systems, Equipment, Munitions, & Computer Programs. This was the first standard that recognized CM of both hardware and software. Even though the defense industry, through various associations, such as the Electronics Industries Association (EIA), the Aerospace Industries Association (AIA), the National Security Industrial Association (NSIA), and the American Electronics Association

(AEA) were reviewing the military standards at that time, it was not until 1988 that commercial standards began to appear.

The proliferation of military standards had a number of drawbacks, and it has been argued that the large number of standards, nearly 30,000 by 1990, imposed unnecessary restrictions and increased costs on contractors and impeded the incorporation of the latest technology. In 1988, the assistant secretary of defense for acquisition, Dr. Costello, wrote a memo indicating that the government should get out of the standards-writing business and entrust the job of developing standards to the organizations that were developing standards on various topics. He also stated that the military would use the standards written by organizations like EIA, IEEE, the Society of Automotive Engineers (SAE), ANSI, and ISO for procuring materials from the commercial market. These organizations had a good track record at developing standards. For example, EIA had written many standards on electronics, electrical, and communications protocols. SAE had developed standards on automotive development and related topics. IEEE was one of the pioneers in the development of software standards. Thus, commercial standards on CM from these organizations began to appear.

In 1994, responding to heightened criticism of the increasing number of military standards, U.S. Secretary of Defense William Perry issued a memorandum that prohibited the use of most defense standards without a waiver, and many defense standards were subsequently canceled. In their place, the DOD encouraged the use of industry standards. Military systems were then required to use "performance specifications" that described the desired features of the weapon, as opposed to requiring a large number of defense standards.

Now, there are hundreds of CM standards available, covering every aspect of CM. A search by the author at the site of Global Engineering Documents, one of the largest vendors of technical standards, for CM standards brought up 151 standards. Some of the popular military and commercial standards are listed as follows.

- DOD-STD-2167A—Defense System Software Development (canceled);
- DOD-STD-2168—Defense System Software Quality Program (canceled);
- MIL-STD-973—Configuration Management, 1990 (canceled);
- MIL-STD-498—Software Development and Documentation, 1994 (canceled);
- MIL-HDBK-61A (SE)—Military Handbook for Configuration Management Guidance, 2001;
- MIL-STD-2549—Configuration Management Data Interface, 1997;
- MIL-STD-480B—Configuration Control Engineering Changes, Deviations and Waivers, 1988 (canceled);
- MIL-STD-481B—Configuration Control Engineering Changes (Short Form), Deviations and Waivers, 1988 (canceled);
- MIL-STD-482—Configuration Status Accounting Data Elements & Related Features, 1974 (canceled);
- MIL-STD-483A—Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs, 1985 (superseded by MIL-STD-973);
- MIL-STD-973—Configuration Management, 1992 (superseded by EIA-649);
- MIL-STD-490A—Specification Practices, 1985 (superseded by MIL-STD-961E);

- MIL-STD-1521B—Technical Reviews and Audits for Systems, Equipment and Computer Programs, 1985 (canceled);
- MIL-STD-961E—Defense and Program-Unique Specifications Format and Content, 2003;
- RTCA DO/178B-92—Software Considerations in Airborne Systems and Equipment Certification;
- NATO STANAG 4159—Configuration Management, 1992;
- STANAG 4427 Ed. 2—Introduction of Allied Configuration Management Publications (ACMPs), 2007;
- ACMP-1 Ed. 2—NATO Requirements for the Preparation of Configuration Management Plans, 2007;
- ACMP-2 Ed. 2—NATO Requirements for Configuration Identification, 2007;
- ACMP-3 Ed. 2—NATO Requirements for Configuration Control—Engineering Changes, Deviations and Waivers, 2007;
- ACMP-4 Ed. 2—NATO Requirements for Configuration Status Accounting and Configuration Data Management, 2007;
- ACMP-5 Ed. 2—NATO Requirements for Configuration Audits, 2007;
- ACMP-6 Ed. 2—NATO Configuration Management Terms and Definitions, 2007;
- ACMP-7 Ed. 2—NATO Configuration Management Guidance on the Application of ACMPs 1 to 6, 2007;
- NATO NAT-PRC-2—Software Project Configuration Management Procedures;
- UK MOD DEF-STAN 05-57/2—Configuration Management Policy and Procedures for Defense Material, 1985;
- NASA-Sfw-DID-04—Software Configuration Management Plan Data Item Description, NASA, 1986;
- NASA D-GL-11—Software Configuration Management for Project Managers, NASA, 1987;
- IEEE-Std-24765-2010—Systems and Software Engineering—Vocabulary, 2012;
- IEEE Std-828-2012—IEEE Standard for Configuration Management in Systems and Software Engineering, 2012;
- ANSI/IEEE Std-1042-1987—IEEE Guide to Software Configuration Management, 1987 (withdrawn);
- ANSI/IEEE Std-730-2014—IEEE Standard for Software Quality Assurance Processes, 2014;
- ANSI/IEEE Std-730.1-1995—IEEE Guide for Software Quality Assurance Planning, 1995 (withdrawn);
- ANSI/IEEE Std-1028-2008—IEEE Standard for Software Reviews and Audits, 2008;
- ISO/IEC/IEEE 12207-2008—ISO/IEC/IEEE Standard for Systems and Software Engineering—Software Life Cycle Processes, 2008;
- ISO/IEC/IEEE 15288-2008—Systems and Software Engineering—System Life Cycle Processes, 2008;
- ISO 9001: 2008—Quality Management Systems—Requirements, 2008;
- ISO 9000-3—Guidelines for the Application of ISO 9001 to the Development and Maintenance of Software, 1997 (withdrawn);

- ISO/IEC 90003:2004—Software Engineering—Guidelines for the Application of ISO 9001:2000 to Computer Software, 2004;
- ISO 10007—Quality Management—Guidelines for Configuration Management, 2003.
- ISO/IEC TR 15846:1998—Information Technology—Software Life Cycle Processes–Configuration Management, 1998;
- EIA-649-B—Configuration Management Standard, 2011;
- EIA-836-B—Configuration Management Data Exchange and Interoperability, 2010;
- EIA CMB4-1A-84—Configuration Management Definitions for Digital Computer Programs, 1984;
- EIA CMB4-2-81—Configuration Identification for Digital Computer Programs, 1981;
- EIA CMB4-3-81—Computer Software Libraries, 1981;
- EIA CMB4-4-82—Configuration Change Control for Digital Computer Programs, 1982;
- EIA CMB5-A-86—Configuration Management Requirements for Subcontractors/Vendors, 1986;
- EIA CMB6-1C-94—Configuration and Data Management References, 1994;
- EIA CMB6-2-88—Configuration and Data Management In-house Training Plan, 1988;
- EIA CMB6-3-91—Configuration Identification, 1991;
- EIA CMB6-4-91—Configuration Control, 1988;
- EIA CMB6-5-88—Textbook for Configuration Status Accounting, 1988;
- EIA CMB6-6-96—Textbook for Reviews and Configuration Audits, 1996;
- EIA CMB6-8-88—Data Management In-house Training Course, 1988;
- EIA CMB6-9-90—Configuration and Data Management Training Course, 1990;
- EIA CMB7-1-91—Electronic Interchange of Configuration Management Data, 1991;
- EIA CMB7-2-91—Guidelines for Transitioning Configuration Management to an Automated Environment, 1991;
- ESA PPS-05-09 Rev. 1—Guide to Software Configuration Management, March, 1995;
- FAA-STD-021 (Rev. A) (Chg Notice 1)—Configuration Management (Contractor Requirements), 1990;
- FEI-4—Software Configuration Management, 1983;
- NIST S.P. 500-161—Software Configuration Management—An Overview;
- BS 6488-84—Code of Practice for Configuration Management of Computer-based Systems;
- JPL D-4011—Software Configuration Management Planning, December 1988.

As indicated in their descriptions, some of the above listed standards (mainly MIL and DOD standards) have been canceled and cannot be used or referred to in any formal agreement or contracts. They are referenced here because they contain valuable information. Moreover, most of them are available on the Internet free of

cost. The following sections provide an overview of a few representative standards from the list—both military and commercial.

Military Standards

The standards published by the U.S. Department of Defense are used by all NATO countries and by countries that use military equipment, manufactured in the United States. The next few sections discuss the following CM-related military standards:

- DOD-STD-2167A—Defense System Software Development (canceled);
- DOD-STD-2168—Defense System Software Quality Program (canceled);
- MIL-STD-973—Configuration Management (canceled);
- MIL-STD-498—Software Development and Documentation (canceled);
- MIL-HDBK-61A (SE)—Military Handbook for Configuration Management Guidance;
- MIL-STD-2549—Configuration Management Data Interface;
- MIL-STD-480B—Configuration Control Engineering Changes, Deviations and Waivers (canceled);
- MIL-STD-481B—Configuration Control Engineering Changes (Short Form), Deviations and Waivers (canceled);
- MIL-STD-482—Configuration Status Accounting Data Elements & Related Features (canceled);
- MIL-STD-973—Configuration Management, (Superseded by EIA-649);
- MIL-STD-1521B—Technical Reviews and Audits for Systems, Equipment and Computer Programs (canceled);
- MIL-STD-961E—Defense and Program-Unique Specifications Format and Content;

DOD-STD-2167A

This standard (which supersedes the DOD-STD-2167—Defense System Software Development, 1985) establishes requirements to be applied during the acquisition, development, or support of software systems. The requirements of this standard apply to the development of CSCIs. Even though this standard was developed for the DOD environment, it can be tailored to handle rapidly evolving software technology and to accommodate a wide variety of state-of-the-practice software engineering techniques. The standard allows the user to incorporate the SCM plan into the software development plan (SDP) or to treat it as a separate document. The benefit of handling the SCM plan as part of the SDP is that, for the projects where SCM is either tightly tied to development life cycle or where the SCM function is relatively small, it allows the SCM plan to be placed in the SDP where it is more appropriate.

DOD-STD-2168

This standard contains requirements for the development, documentation, and implementation of a software quality program. This program includes planning for and

conducting evaluations of the quality of software, associated documentation, and related activities and planning for and conducting the follow-up activities necessary to assure timely and effective resolution of problems. This standard, together with other military specifications and standards governing software development, CM, specification practices, project reviews and audits, and subcontractor management, provides a means for achieving, determining, and maintaining quality in software and associated documentation.

MIL-STD-498

The purpose of this standard was to establish uniform requirements for software development and documentation. This standard merged DOD-STD-2167A, DOD-STD-7935A, and DOD-STD-1703 to define a set of activities and documentation suitable for the development of both weapon systems and automated information systems. This standard supersedes DOD-STD-2167A, DOD-STD-7935A, and DOD-STD-1703.

A conversion guide from these standards to MIL-STD-498 is provided in Appendix A. Other changes include improved compatibility with incremental and evolutionary development models, improved compatibility with nonhierarchical design methods; improved compatibility with computer-aided software engineering (CASE) tools; alternatives to, and more flexibility in, preparing documents; clearer requirements for incorporating reusable software; introduction of software management indicators; added emphasis on software supportability; and improved links to systems engineering. This standard is superseded by IEEE/EIA 12207.0, IEEE/EIA 12207.1 and IEEE/EIA 12207.2

MIL-HDBK-61A (SE)

This military handbook provides guidance and information to DOD acquisition managers, logistics managers, and other individuals assigned responsibility for CM. It helps them plan for and implement effective DOD CM activities and practices during all life cycle phases of defense systems and CIs. It supports acquisition based on performance specifications, and the use of industry standards and methods to the greatest practicable extent. Revision B of this document is currently in draft form and is being reviewed.

This military handbook's content is structured to provide a comprehensive guide (roadmap). After the initial three sections—Scope, Applicable Documents and Definitions—the handbook is divided into the following major sections:

- CM Life Cycle Management and Planning (Section 4)—Since management and planning are the keys to effective implementation of CM, Section 4 provides the focus for the entire handbook. It contains an overview of the CM process, a discussion of CM's relationships to other processes, and a synopsis of government/contractor CM during the entire program life cycle. It addresses global CM activities applicable to all phases such as planning, process implementation, and performance measurement.

- Major CM Functions (Sections 5–9.)—In support of Section 4, Sections 5–9 contain detailed information in the form of activity descriptions, activity models, principles and concepts, and activity guides (e.g., diagrams, checklists, and tables) for the following topics: configuration identification, configuration control, CSA, configuration verification, and audit and data management.

Appendix F of this handbook contains a comparison of the CM standards such as ISO 10007, IEEE Std-828, and MIL-STD-973 against EIA-649. This matrix is very useful as it gives the relative strengths and weaknesses of the four CM standards.

MIL-STD-2549

This document establishes a standard interface for the delivery of, or access to, electronic CM data. This interface prescribes the data elements, the data element definitions, and the data element relationships that define the conceptual schema for CM data. These interface requirements have been subdivided into data information packets to support various CM needs. This standard applies to all activities responsible for procuring, recording, maintaining, and disseminating CM information.

MIL-STD-480B

This standard establishes the requirements, formats, and procedures to be utilized in the preparation of configuration control documentation. Included are requirements for the following:

- Maintaining configuration control of CIs, both hardware and software;
- Preparing and submitting engineering change proposals (ECPs), requests for deviations (RFDs)/requests for waivers (RFWs), notices of revision (NORs), and SCNs;
- Evaluating, coordinating, and approving or disapproving ECPs and RFDs/RFWs applicable to the DOD—NDIs or commercial items.

This standard establishes configuration control requirements and procedures applicable to the acquisition and modification of items procured by the DOD. This standard is to be used by contractors and government activities to do the following:

- Establish and maintain effective configuration control of the approved configuration identification;
- Propose engineering changes to CIs, both hardware and software, that are designed, developed, or modified for DOD activities;
- Request deviations or waivers pertaining to such items;
- Prepare NORs and SCNs;
- Control the form, fit, and function of privately developed items used in CIs, including NDI items.

MIL-STD-481B

This standard establishes requirements, formats, and procedures for the preparation, submission, and approval or disapproval of abbreviated ECPs. Where complete descriptions of ECPs are required, MIL-STD-480 should be specified in contracts. The purpose of this standard is to establish configuration control requirements and procedures applicable to the acquisition and modification of items procured by the DOD. It is intended that this standard be applied to contracts or orders for procurement of the following:

- Multiapplication or standard items that were not developed as subdivisions of a specific system;
- Items fabricated in accordance with a mandatory detail design that was not developed by the fabricator;
- Privately developed items (e.g., COTS items), when the procuring activity has determined that the application of change control to such items is necessary and that the short form ECP is applicable.

MIL-STD-482

To assure the use of uniform, clearly defined status-accounting management information throughout the DOD and the DOD-defense Industry interface, this standard prescribes status-accounting standard data elements, interim (nonstandard) data elements, and their related data items, codes, use identifiers, and data chains (referred to as "related features"). The data elements and related features are to be used as the content of those CSA records prepared by or for the department or agencies of DOD in accordance with the provisions of DOD Directive 5010.19 and DOD Instruction 5010.21.

MIL-STD-973

This standard defines CM requirements that are to be selectively applied, as required, throughout the life cycle of any CI that fits either of the following descriptions:

- Developed wholly or partially with government funds, including NDIs when the development of technical data is required to support off-the-shelf equipment of software;
- Designated for CM for reason of integration, logistics support, or interface control.

This standard is superseded by EIA-649.

MIL-STD-1521B

This standard, which supersedes the MIL-STD-1521 (Technical Reviews & Audits for Systems, Equipment, & Computer Software), prescribes the requirements for the conduct of technical reviews and audits on systems, equipment, and computer

software. The program manager shall select the following technical reviews and audits at the appropriate phase of program development:

- System requirements review (SRR);
- System design review (SDR);
- Software specification review (SSR);
- Preliminary design review (PDR);
- Critical design review (CDR);
- TRR;
- FCA;
- PCA;
- Formal qualification review (FQR);
- Production readiness review (PRR).

Technical reviews and audits defined in this standard are to be conducted in accordance with this standard to the extent specified in the contract clauses, statement of work (SOW), and the contract data requirements list.

MIL-STD-961E

This standard, which supersedes MIL-STD-490A, establishes the format and content requirements for the preparation of defense specification and program-unique specifications prepared either by DOD activities or by contractors for the DOD.

International/Commercial Standards

There are a host of standards by many organizations like the EIA, the Electric Power Research Institute (EPRI), the European Computer Manufacturers Institute (ECMI), the Federal Aviation Authority (FAA), the Institute of Nuclear Power Operations (INPO), the European Space Agency (ESA), the Nuclear Information & Records Management Association (NIRMA), NASA, and North Atlantic Treaty Organization (NATO). However, the usage of these standards is limited to the members of those organizations. The most popular international standards on CM are those by ANSI/IEEE and ISO. The next sections will describe the following popular international and commercial standards:

- EIA-649-B—Configuration Management Standard;
- IEEE Std-828-2012—IEEE Standard for Configuration Management in Systems and Software Engineering;
- ANSI/IEEE Std-1042-1987—IEEE Guide to Software Configuration Management;
- ANSI/IEEE Std-730-2014—IEEE Standard for Software Quality Assurance Processes;
- ANSI/IEEE Std-730.1-1995—IEEE Guide for Software Quality Assurance Planning;
- ANSI/IEEE Std-1028-2008—Standard for Software Reviews and Audits;

- ISO/IEC/IEEE 12207-2008—ISO/IEC/IEEE Standard for Systems and Software Engineering—Software Life Cycle Processes;
- ISO/IEC/IEEE 15288-2008—Systems and Software engineering—System life cycle processes;
- ISO 9001: 2008—Quality Management Systems—Requirements;
- ISO/IEC 90003:2004—Guidelines for the Application of ISO 9001:2000 to Computer Software;
- ISO 10007: 2003—Quality management—Guidelines for Configuration Management.

EIA-649-B

This standard defines five CM functions and their underlying principles. The principles, highlighted in text boxes, are designed to individually identify the essence of the related CM functions and can be used collectively to create a checklist of criteria to evaluate a CM program. In describing each CM function and its principles, this standard utilizes neutral CM terminology, while also providing equivalent terms that have historically been used in various product environments. There is no intent to express preference for any particular set of terminology. This standard uses a neutral set of names for the phases of a product's life cycle, which are generic enough to be easily mapped to the myriad of different life cycle models in use. Regardless of the titles chosen for the various life cycle phases, or whether the product is a facility, software, an airplane, or a machine screw, at some time in its history a product will go through all or most of these phases. The phases can have considerable overlap, or the sequence of the phases might change or be repeated (e.g., for product improvements and enhancements). Approved configurations of a product can be in the build, distribution, operation, and disposal phases simultaneously, and changes to those configurations may occur during all life cycle phases. Appropriate application of CM functions enables a user of this standard to plan and implement a CM program for a product, project, or enterprise. The degree to which each of the CM principles applies to a product varies over the product's life cycle. Some principles do not apply during every phase of the product's life cycle (e.g., configuration verification and audit principles are not applicable in the conception or definition phases). The degree of rigor and techniques used in implementing CM is commensurate with the type of product and its application environment as defined by program requirements.

IEEE Std-828-2012

This standard establishes the minimum requirements for processes for CM in systems and software engineering. The application of this standard applies to any form, class, or type of software or system. This revision of the standard expands the previous version to explain CM, including identifying and acquiring CIs, controlling changes, reporting the status of CIs, as well as software builds and release engineering. Its predecessor defined only the contents of a software CM plan. This standard addresses what CM activities are to be done, when they are to happen in the life cycle, and what planning and resources are required. It also describes the

content areas for a CM plan. The standard supports ISO/IEC/IEEE 12207:2008 (Standard for Systems and Software Engineering—Software Life Cycle Processes) and ISO/IEC/IEEE 15288:2008 (Systems and Software Engineering System Life Cycle Processes) and adheres to the terminology in ISO/IEC/IEEE Std. 24765 and the information item requirements of IEEE Std. 15939.

ANSI/IEEE Std-1042-1987

This is the most comprehensive international standard available on SCM. This standard describes the application of CM disciplines to the management of software engineering projects. SCM consists of two major aspects—planning and implementation. For those planning SCM activities, this standard provides insights into the various factors that must be considered. Users implementing SCM disciplines will find suggestions and detailed examples of SCM plans in this standard. This standard introduces the essential concepts of SCM, particularly those of special significance (for example, libraries and tools) to software engineering. It then presents the planning for SCM in terms of documenting a plan following the outline of the ANSI/IEEE Std-828, so that a user who is unfamiliar with the disciplines of SCM can gain valuable insights into the issues. For those preparing SCM plans, the second part of the guide provides sample plans for consideration.

ANSI/IEEE Std-730-2014

Requirements for initiating, planning, controlling, and executing the SQA processes of a software development or maintenance project are established in this standard. This standard is harmonized with the software life cycle process of ISO/IEC/IEEE 12207:2008 and the information content requirements of ISO/IEC/IEEE 15289:2011.

ANSI/IEEE Std-730.1-1995

This guide explains and clarifies the contents of each section of a SQA plan (SQAP) that satisfies the requirements of IEEE Std-730-1989. The guide supersedes IEEE Std-983-1986 and does not constitute further requirements than those stated in IEEE Std-730-1989. An organization can claim compliance with IEEE Std-730-1989 without following this guide completely. This guide presents the consensus of those in the software development and maintenance community with expertise or experience in generating, implementing, evaluating, and modifying SQAPs. The SQAP should describe the plans and activities for the SQA staff. The SQA staff observes the development process and reports deficiencies observed in the procedures and the resulting products.

ANSI/IEEE Std-1028-2008

This standard provides definitions and uniform requirements for review and audit processes. It does not establish the need to conduct specific reviews or audits; that need is defined by local policy. Where specific reviews and audits are required,

standard procedures for their execution must be defined. This standard provides definitions for review and audit purposes that are applicable to products and processes throughout the software life cycle. Each organization should specify where and when this standard applies and any intended deviations from this standard.

This standard defines five types of software reviews and audits, together with procedures required for the execution of each type. It is concerned only with the reviews and audits; it does not define procedures for determining the necessity of a review or audit, nor does it specify the disposition of the results of the review or audit. Review types include management reviews, technical reviews, inspections, and walk-throughs. This standard is meant to be used either in conjunction with other IEEE software engineering standards or as a stand-alone definition of software review and audit procedures. In the latter case, local management must determine the events that precede and follow the actual software reviews and audits.

ISO/IEC/IEEE 12207-2008

This standard establishes a common framework for software life cycle processes, with well-defined terminology, that can be referenced by the software industry. It contains processes, activities, and tasks that are to be applied during the acquisition of a software product or service and during the supply, development, operation, maintenance, and disposal of software products. Software includes the software portion of firmware.

This standard applies to the acquisition of systems and software products and services, to the supply, development, operation, maintenance, and disposal of software products and the software portion of a system, whether performed internally or externally to an organization. Those aspects of system definition needed to provide the context for software products and services are included. This standard also provides a process that can be employed for defining, controlling, and improving software life cycle processes. The processes, activities and tasks of ISO/IEC/IEEE 12207-2008—either alone or in conjunction with ISO/IEC 15288—may also be applied during the acquisition of a system that contains software.

ISO/IEC/IEEE 15288:2008

This standard establishes a common framework for describing the life cycle of systems created by humans. It defines a set of processes and associated terminology. These processes can be applied at any level in the hierarchy of a system's structure. Selected sets of these processes can be applied throughout the life cycle for managing and performing the stages of a system's life cycle. This is accomplished through the involvement of all interested parties, with the ultimate goal of achieving customer satisfaction. ISO/IEC 15288:2008 also provides processes that support the definition, control, and improvement of the life cycle processes used within an organization or a project. Organizations and projects can use these life cycle processes when acquiring and supplying systems. This standard concerns those systems that are man-made and may be configured with one or more of the following: hardware, software, data, humans, processes (e.g., processes for providing service to users),

procedures (e.g., operator instructions), facilities, materials, and naturally occurring entities. When a system element is software, the software life cycle processes documented in ISO/IEC 12207:2008 may be used to implement that system element. ISO/IEC 15288:2008 and ISO/IEC 12207:2008 are harmonized for concurrent use on a single project or in a single organization.

ISO 9001:2008

ISO 9001:2008, which replaced ISO 9001:2000, is a model for QA when conformance to specified requirements is to be assured by the supplier during design, development, production, installation, and servicing. ISO 9001:2008 specifies requirements for a quality management system where an organization needs to demonstrate its ability to consistently provide product that meets customer and applicable statutory and regulatory requirements and aims to enhance customer satisfaction through the effective application of the system, including processes for continual improvement of the system and the assurance of conformity to customer and applicable statutory and regulatory requirements. This standard specifies quality system requirements for use where a supplier's capability to design and supply conforming product needs to be demonstrated. The requirements specified are aimed primarily at achieving customer satisfaction by preventing nonconformity at all stages from design through to servicing. ISO 9000 is not a standard that is specific to software development. It is a general standard and can be tailored to any industry. This standard does not mention the term CM anywhere in it but contains most CM concepts. A few examples are detailed as follows:

- There should be a mechanism to control quality system documents. This mechanism should approve documents before they are distributed, provide the correct version of documents at points of use, review and reapprove documents whenever they are updated, specify the current revision status of the documents, monitor documents that come from external sources, prevent the accidental use of obsolete documents, and preserve the usability of your quality documents.
- All design changes and modifications shall be identified, documented, reviewed, and approved by authorized personnel before their implementation.
- The documents and data shall be reviewed and approved for adequacy by authorized personnel prior to issue. A master list or equivalent document control procedure identifying the current revision status of documents shall be established.
- Where and to the extent that traceability is a specified requirement, the supplier shall establish and maintain documented procedures for unique identification of individual product or batches. This identification shall be recorded.
- Nonconforming products shall be reviewed in accordance with documented procedures. They may be reworked to meet the specified requirements, accepted with or without repair by concession, regarded for alternative applications, or rejected or scrapped.
- The supplier shall implement and record any changes to the documented procedures resulting from corrective and preventive action.

- The organization should maintain quality system records and should be able to prove that requirements have been met using the records. A procedure to control the records should be developed, and it must be ensured that all the records are useable.
- The organization should manage design and development changes by identifying, recording, reviewing, verifying, validating, and approving the changes in product design and development.
- The organization should identify and track its products by establishing, maintaining, and recording the identity of the products.
- The organization should control nonconforming products by developing a procedure to control nonconforming products, identifying and controlling the nonconforming products, reverifying the nonconforming products that were corrected, controlling nonconforming products after delivery or use, and maintaining records of nonconforming products.
- Organizations must correct actual nonconformities by reviewing the nonconformities, finding out what causes the nonconformities, evaluating whether corrective action is necessary, developing corrective actions to prevent recurrence, recording the results that corrective actions achieve, and examining the effectiveness of corrective actions.

All requirements of ISO 9001:2008 are generic and are intended to be applicable to all organizations, regardless of type, size, and product provided.

ISO/IEC 90003: 2004

The ISO 9000 series of standards were not primarily designed for software but for manufacturing processes. ISO 9001 is the model for QA in design and development, production, installation, and servicing (or, in other words, manufacturing processes, which have design aspects). ISO/IEC 90003:2004 provides guidance for organizations in the application of ISO 9001:2000 to the acquisition, supply, development, operation, and maintenance of computer software and related support services. ISO/IEC 90003:2004 does not add to or otherwise change the requirements of ISO 9001:2000. The guidelines provided in ISO/IEC 90003:2004 are not intended to be used as assessment criteria in quality management system registration or certification.

The application of ISO/IEC 90003:2004 is appropriate to software that is part of a commercial contract with another organization, a product available for a market sector, used to support the processes of an organization, embedded in a hardware product, or related to software services. Some organizations may be involved in all the above activities; others may specialize in one area. Whatever the situation, the organization's quality management system should cover all aspects (software related and nonsoftware related) of the business. ISO/IEC 90003:2004 identifies the issues that should be addressed and is independent of the technology, life cycle models, development processes, sequence of activities, and organizational structure used by an organization.

The standard ISO 9000-3 contains guidelines for the application of ISO 9001 to the development, supply, and maintenance of software. This standard mentions

identification of CM procedures as a part of quality planning, mentions the need to develop a SCM plan, and stresses the importance of bringing the various artifacts under configuration control prior to use. For CM functions and procedures this standards refers to ISO 10007:2003.

ISO 10007: 2003

ISO 10007:2003 gives guidance on the use of CM within an organization. It is applicable to the support of products from concept to disposal. It first outlines the responsibilities and authorities before describing the CM process, which includes CM planning, configuration identification, change control, CSA, and configuration audit. Since ISO 10007:2003 is a guidance document, it is not intended to be used for certification or registration purposes.

ISO 10007 defines CM as a management discipline that applies technical and administrative direction to the development, production, and support life cycle of a CI. The main objective of CM is to document and provide full visibility of the product's present configuration and on the status of achievement of its physical and functional requirements. Another objective is that everyone working on the project at any time in its life cycle uses correct and accurate documentation.

This standard is applicable to the support of projects from concept to design, development, procurement, production, installation, operation, and maintenance and to the disposal of products. The application of CM may be tailored to suit individual projects, taking into account the size, complexity, and nature of the work.

Summary

The ANSI/IEEE standards are the most widely used SCM standards, and the coverage of software CM and its functions is quite elaborate and comprehensive. ISO 9000-3 and ISO 10007 provide another set of very good CM standards. The EIA-649-B is perhaps the most readable SCM standard available. The DOD and MIL standards are mainly used in defense industry projects, and even though they could be tailored for any project their use is generally limited to the defense industry and military organizations within and outside the United States.

The SCM standards (most preferably EIA-649-B) are the starting point for the practice of CM and related functions in any project or organization. It is the first place that one should look for guidance when one is starting a CM program. Unless your organization does not deal in the defense industry, then it would be better to base your SCM system on one of the commercial standards like ANSI/IEEE or ISO. This is because these standards are written for the entire industry (whereas the DOD standards were written for their specific segments of industry), and hence they are more flexible and can be customized more easily to suit your needs. Also, these standards have greater potential for timely updates than the DOD standards; since they are used by the general industry, they must maintain relevance to the current software engineering principles and practices or face obsolescence. The CM standards have played a very crucial role in shaping the way in which CM is being practiced today.

Selected Bibliography

Alain Abran, A., and Moore, J. W., (eds.), *SWEBOK: Guide to the Software Engineering Body of Knowledge (Trial Version)*, Los Alamitos, California: IEEE Computer Society, 2001.

Ben-Menachem, M., *Software Configuration Management Guidebook*, New York: McGraw-Hill, 1994.

Berlack, R.H., *Software Configuration Management*, New York: John Wiley & Sons, Inc., 1992.

IEEE, *IEEE Guide for Software Quality Assurance Planning (IEEE Std-730.1–1995)*, Piscataway, NJ: IEEE, 1995.

IEEE, *IEEE Guide to Software Configuration Management (IEEE Std-1042-1987)*, Piscataway, NJ: IEEE, 1987.

IEEE, *IEEE Standard for Configuration Management in Systems and Software Engineering (IEEE Std-828–2012)*, Piscataway, NJ: IEEE, 2012.

IEEE, *IEEE Standard for Software Quality Assurance Plans (IEEE Std-730–1998)*, Piscataway, NJ: IEEE, 1998.

IEEE, *IEEE Standard for Systems and Software Engineering—Software Life Cycle Processes (IEEE Std-12207-2008)*, Piscataway, NJ: IEEE, 2008.

IEEE, *Standard for Software Reviews and Audits (IEEE Std-1028-2008)*, Piscataway, NJ: IEEE, 2008.

IEEE, *Systems and Software Engineering—System life cycle processes (IEEE Std-15288-2008)*, Piscataway, NJ: IEEE, 2008.

Ince, D., *ISO 9001 and Software Quality Assurance*, London: McGraw-Hill Book Company, 1994.

ISO, *Guidelines for the Application of ISO 9001:2000 to Computer Software (ISO 90003:2004)*, Geneva: ISO, 2004.

ISO, *Quality Management Systems—Requirements (ISO 9001:2008)*, Geneva: ISO, 2008.

ISO, *Quality Management—Guidelines for Configuration Management (ISO 10007:2003)*, Geneva: ISO, 2003.

Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

Peach, R. W. (ed.), *The ISO 9000 Handbook*, New York: The McGraw-Hill Companies, Inc., 1997.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

TechAmerica, *Configuration Management Standard ANSI/EIA-649-B*, Washington, D.C., 2011.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

CHAPTER 12

Software Process Improvement Models and SCM

Introduction

A process is a series of interrelated activities that bring about a result or that are directed toward a particular aim. Process improvement is the analysis and redesign of processes to eliminate organizational problems and inefficiencies in small increments, over time, by improving one or two processes at a time. Process improvement is done on an operational level (as opposed to radical reengineering, which is done on a strategic level) and is carried out primarily by the people most involved in the process. The incremental process improvement approach builds in small successes that motivate teams to continue. Failure, if it occurs, has less potential to do serious damage because scope is limited to one or two processes at a time.

Process improvement models define the various processes and activities that will help an organization to move from a stage of total confusion, lack of discipline, and ad hoc processes to a stage where the organization has well-defined and mature processes that help the organization not only to deliver high-quality products and services repeatedly but also to continuously improve the quality of the products and services and efficiency of the organization.

The software process improvement models assess the current state of an organization and determine where it belongs in the process maturity scale. Then these models give specific guidelines to move from the current state to a process maturity level where the organizations have the necessary processes to continuously improve its products and services.

This section describes some of the most common process assessment and improvement models and standards. These include CMM, CMMI, ISO/IEC 15504 (formerly SPICE), BOOTSTRAP, Trillium, ITIL, COBIT, and SWEBOK. We will examine the role of CM in process improvement and how the above-mentioned process improvement and assessment models relate CM and process improvement.

CMM

The CMM for software describes the principles and practices underlying software process maturity and is intended to help software organizations improve the maturity of their software processes in terms of an evolutionary path from ad hoc, chaotic processes to mature, disciplined software processes. The CMM is organized

into five maturity levels—initial, repeatable, defined, managed, and optimizing. A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.

Except for level 1, each maturity level is decomposed into several key process areas that indicate the areas an organization should focus on to improve its software process. Key process areas identify the issues that must be addressed to achieve a maturity level. Each key process area identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability. CM is a key process area for level 2. So for any organization that wants to achieve CMM level 2 or above, CM practices have to be performed in accordance with the guidance provided by the CMM document.

According to the CMM documents, the CM practice should achieve the following four goals:

- Software CM activities are planned;
- Selected software work products are identified, controlled, and available;
- Changes to identified software work products are controlled;
- Affected groups and individuals are informed of the status and content of software baselines.

To achieve the above goals the CMM requires certain commitments from the organization—commitment to perform. The organizations should also have the necessary resources (e.g., funding, tools, and training capabilities) to achieve these goals—ability to perform. The CMM also specifies several activities to be performed to achieve the goals—activities performed. The model also provides for mechanisms for measuring, and reviewing data—measurement and analysis, and for analyzing, statusing, and auditing—verifying implementation.

CMM Interactive (CMMI)

The CMMI project was formed to address the need that computer system development environments were concerned with more than just software. The CMMI product team's mission was to combine three source models:

- The CMM for Software (SW-CMM) v2.0 draft C;
- The Systems Engineering Capability Model (SECM);
- The Integrated Product Development Capability Maturity Model (IPD-CMM) v 0.98.

The combination of these models into a single improvement framework is intended for use by organizations in their pursuit of enterprise-wide process improvement.

There are two types of CMMI model representations—staged and continuous. The staged representation is the approach used in the SW-CMM. It is an approach that uses predefined sets of process areas to define an improvement path for an organization. This improvement path is described by a model component called

a maturity level. A maturity level is a well-defined evolutionary plateau toward achieving improved organizational processes for a project (i.e., the project moves through the maturity levels).

The continuous representation allows an organization to select a specific process area and improve relative to it. The continuous representation uses capability levels to characterize improvement relative to an individual process area. A capability level is a well-defined evolutionary plateau toward achieving improved organizational processes for a particular organization. An organization moves through the capability levels until it reaches the top.

CM comes under the "support" process area of CMMI. CMMI defines CM as a discipline whose purpose is to establish and maintain the integrity of work products using configuration identification, configuration control, CSA, and CAs.

The CM process area supports all process areas by establishing and maintaining the integrity of work products using configuration identification, configuration control, CSA, and CAs. The work products placed under CM include the products that are delivered to the customer, designated internal work products, acquired products, tools, and other items that are used in creating and describing these work products. Examples of work products that may be placed under CM include plans, process descriptions, requirements, design data, drawings, product specifications, code, compilers, product data files, and product technical publications.

The specific goals for CM as defined in CMMI are listed as follows:

- *Establish baselines:* Baselines of identified work products are established.
- *Track and control changes:* Changes to the work products under CM are tracked and controlled.
- *Establish integrity:* Integrity of baselines is established and maintained.

The specific goals are achieved by following a number of specific practices. The practices for each specific goal of CM are:

- Establish baselines (SG 1):
  1. Identify configuration items (SP 1.1-1).
  2. Establish a CM System (SP 1.2-1).
  3. Create or release baselines (SP 1.3-1).
- Track and control changes (SG 2):
  1. Track change requests (SP 2.1-1).
  2. Control configuration items (SP 2.2-1).
- Establish integrity (SG 3):
  1. Establish CM records (SP 3.1-1).
  2. Perform CAs (SP 3.2-1).

ISO/IEC 15504

ISO/IEC 15504 is a process improvement and assessment standard. This standard was developed jointly by the Joint Technical Committee 1/Sub Committee (JTC1/SC7) of the ISO and International Electrotechnical Commission (IEC). The work

was assigned to a group called WG10, which established a special project called Software Process Improvement and Capability dEtermination (SPICE). The SPICE project, established in 1993, developed a set of draft standards that finally (after countless reviews and revisions) evolved into the ISO/IEC 15504 standard.

The ISO/IEC 15504 process model is divided into five categories—organizational process (ORG), management process (MNA), customer-supplier process (CUS), engineering process (ENG), and support process (SUP). Each of these processes is divided into several process areas. CM comes under the support process—SUP. 2.

The purpose of SUP.2 is to establish and maintain the integrity of all of the work products of a process or project. The successful implementation of the process should lead to the following outcomes:

- Identifying, defining, and baselining all relevant items generated by the process or project;
- Controlling modifications and releases;
- Recording and reporting the status of the items and modification requests;
- Ensuring the completeness and consistency of the items;
- Controlling storage, handling, and delivery of the items.

The standard defines nine best practices that will lead to the successful implementation of CM system. These include developing a CM strategy, establishing a CM system, CI identification, maintaining the CI description and history, establishing formal change management procedures, managing product releases, maintaining the CI history, reporting configuration status, and managing the release and delivery of CIs.

This model has six levels (0–5) of process capability—incomplete, performed, planned and tracked, established, predictable, and optimizing. Since this model is continuous, the CM must be performed at the different levels. To obtain level 1, CM must be performed in such a way that the expected outcomes are achieved. Level 2 requires placing the work products under configuration control and developing a CM plan. To obtain level 3, the CM procedures should be documented and the resources required for performing CM must be made available. Level 4 requires establishing metrics for measuring the performance of CM and controlling the performance based on the measurements. At level 5, the CM metrics are used to improve the CM performance and again used to measure this improved performance.

BOOTSTRAP

BOOTSTRAP is a method to evaluate and to improve the quality of the software development and management processes of an organization. BOOTSTRAP is based on the assumption that the capabilities of an organization increase one step after the other. The concepts behind this approach were published by Watts Humphrey while working for the Software Engineering Institute (SEI). The CMM distinguishes five levels of process quality, while the CMMI has six levels. This model is now the software process standard in the United States. BOOTSTRAP uses the same basic

principles. The reference model has been extended cover the requirements imposed by ISO 9000. BOOTSTRAP does not only deliver the maturity level, but also a detailed capability profile of the organization and the projects investigated.

BOOTSTRAP adopts a process model that addresses the processes and practices for the organization and the project. The process areas are categorized into organization, methodology, and technology. The process categories are comprised of process areas that contain activities and best practices. The methodology process is again divided into three process areas—process engineering, product engineering, and engineering support functions. Each of these process areas contains various activities. Configuration and change management is an activity under engineering support functions. The BOOTSTRAP process model contains many best practices for the configuration and change management activity. These are very similar to those of ISO/IEC 15504.

The BOOTSTRAP model has five maturity levels—initial, repeatable, defined, managed, and optimizing. The process areas are not confined to a single level but cover several levels.

Trillium Model

The goal of the Trillium Model is to provide a means to initiate and guide a continuous improvement program. The model is used in a variety of ways:

- To benchmark an organization's product development and support process capability against best practices in the industry;
- In self-assessment mode, to help identify opportunities for improvement within a product development organization;
- In precontractual negotiations, to assist in selecting a supplier.

This model and its accompanying tools are not in themselves a product development process or life cycle model. Rather, the Trillium Model provides key industry practices that can be used to improve an existing process or life cycle. The practices in the Trillium Model are derived from a benchmarking exercise that focused on all practices that would contribute to an organization's product development and support capability. Trillium has a telecommunications orientation; provides a customer focus; provides a product perspective; covers CMM, ISO 9001, ISO 9000-3, Bellcore's TR-NWT-000179 and TA-NWT-001315, Malcom Baldrige National Quality Award criteria, IEEE, and IEC standards; includes technological maturity; includes additional Trillium-specific practices; and provides a roadmap approach that sequences improvements by maturity.

The Trillium Model has been developed from a customer perspective, as perceived in a competitive, commercial, environment. In this context, capability is defined as the ability of a development organization to consistently deliver a product or an enhancement to an existing product that meets customer expectations with minimal defects for the lowest life cycle cost, and in the shortest time. A telecommunications product typically includes hardware, software, documentation, training, and support services.

The Trillium scale spans levels 1 through 5. The levels can be characterized in the following way:

1. *Unstructured:* The development process is ad hoc. Projects frequently cannot meet quality or schedule targets. Success, while possible, is based on individuals rather than on organizational infrastructure (risk = high).
2. *Repeatable and project-oriented:* Individual project success is achieved through strong project management planning and control, with emphasis on requirements management, estimation techniques, and CM (risk = medium).
3. *Defined and process-oriented:* Processes are defined and used at the organizational level, although project customization is still permitted. Processes are controlled and improved. ISO 9001 requirements such as training and internal process auditing are incorporated (risk = low).
4. *Managed and integrated:* Process instrumentation and analysis is used as a key mechanism for process improvement. Process change management and defect prevention programs are integrated into processes. CASE tools are integrated into processes (risk = lower).
5. *Fully integrated:* Formal methodologies are extensively used. Organizational repositories for development history and process are utilized and effective (risk = lowest).

Each level (except level 1) contains several capability areas, which, in turn, contain roadmaps that are comprised of activities. This is very similar to the CMM. Each capability area spans multiple levels. For example, the development practices capability area spans levels 2, 3, and 5. The development practices capability area contains the CM roadmap among another six roadmaps (development process, development techniques, internal documentation, verification and validation, reuse, and reliability management). The configuration management related activities of the Trillium model are shown in Table 12.1.

Information Technology Infrastructure Library (ITIL)

ITIL is a globally recognized collection of best practices for information technology (IT) service management. It provides a practical framework for identifying, planning, delivering, and supporting IT services to the business. United Kingdom's Central Computer and Telecommunications Agency (CCTA) created ITIL in response to growing dependence on information technology for meeting business needs and goals.

ITIL provides businesses with a customizable framework of best practices to achieve quality service and overcome difficulties associated with the growth of IT systems.

The primary objective of service management is to ensure that IT services are aligned with the business needs and actively support them. It is important that IT acts as an agent for change to facilitate business transformation. All organizations that use IT depend on IT to be successful. If IT processes and IT services are implemented, managed, and supported in the appropriate way, the business will be more successful, suffer less disruption and loss of productive hours, reduce costs, increase revenue, improve public relations, and achieve its business objectives.

Table 12.1 CM in Trillium Model

Level	*Criteria*	*Description*
2	Scope	Source code is under CM control
		All project and product (internal and external) documents are under CM control
	Function	A board having the authority for managing the project's product baselines [i.e., a product configuration control board (PCCB)] exists or is established
		There is a function responsible for coordinating and implementing CM for the project
	Funding	Adequate resources and funding are provided for performing the CM activities
	Planning	A CM plan is prepared for each project according to a documented procedure
		A documented and approved CM plan is used as the basis for performing the CM activities
	Repository	A CM system is established as a repository for product baselines
		The product repository ensures secure storage of configuration items (e.g., code units and design documents) and the secure and controlled retrieval of current and previous versions of CIs
		The product repository ensures the secure and controlled retrieval of current and previous baselines
		The status of CIs/units is recorded according to a documented procedure
		The product repository maintains records of the status and change history of all CIs and baselines
	Traceability	There is traceability between design specifications and source code and between design specifications and integration test cases
	Change control	CRs and PRs for all configuration items/units are initiated, recorded, reviewed, approved, and tracked according to a documented procedure
		CIs and baselines are changed formally according to a documented procedure
	Baselines	Baseline(s) are created and released formally
		Product baseline audits are conducted according to a documented procedure
	Reporting	Standard reports documenting the CM activities and the contents of the product baselines are developed and made available to affected groups and individuals
3	Scope	Plans, descriptions, product test procedures, requirements specifications, design specifications, review results, and test cases (e.g., integration, system and operation) are under CM control
		All development tools are under CM control
	Traceability	There is full forward and backward traceability between all configuration items (e.g., design specification forward to code units and design specification backward to requirement specification)
5	Scope	The complete development history (e.g., design decisions and design rationale) is captured and maintained under CM control

ITIL provides guidance throughout the service life cycle to help senior business managers and IT managers achieve the objectives of service management and address the key issues they face in a systematic way.

ITIL guidance is structured in five life cycle phases—service strategy, service design, service transition, service operation, and continual service improvement. The SCM processes are under service transition stage. These processes, described in subsequent sections, are listed as follows:

- Change evaluation;
- Change management;
- Release and deployment management;
- Service asset and CM.

Change Evaluation

The purpose of the change evaluation process is to provide a formal, standardized means of determining the performance of a service change in the context of likely impacts on business outcomes, and on existing and proposed services and IT infrastructure. Change evaluation assesses the actual performance of a change against its predicted performance and identifies risks and issues related to the change. Change evaluation is closely linked to change management. The main output of change evaluation is an evaluation report, which is used to help change management personnel decide whether to authorize a change. Formal change evaluation is not required for all changes, and each service provider defines when this formal process should be used and when the evaluation can be carried out as part of change management. The main subprocesses of change evaluation are change evaluation prior to planning, change evaluation prior to build, change evaluation prior to deployment, and change evaluation after deployment.

Change Management

Change management controls the life cycle of all changes, enabling beneficial changes to be made with minimum disruption to IT services. Change management ensures that changes are recorded and evaluated and that authorized changes are prioritized, planned, tested, implemented, documented, and reviewed in a controlled manner. The main subprocesses of the change management process are change management support, assessment of change proposals, request for change (RFC) logging and review, assessment and implementation of emergency changes, change assessment by the change manager, change assessment by the change advisory board (CAB), change scheduling and build authorization, change deployment authorization, minor change deployment, post-implementation review, and change closure.

Release and Deployment Management

The purpose of the release and deployment management process is to plan, schedule, and control the building, testing, and deployment of releases and to deliver new functionality required by the business while protecting the integrity of existing services.

Effective release and deployment delivers significant business value by delivering changes at optimized speed, risk, and cost and offering a consistent, appropriate, and auditable implementation of usable and useful services. Release and deployment management covers the whole build, test, and implementation of new or changed services, from planning through to early life support. The subprocesses of this phase are release management support, release planning, release build, release deployment, early life support, and release closure.

Service Asset and CM

The purpose of service asset and CM (SACM) is to ensure that the assets required to deliver services are properly controlled, and that accurate and reliable information about those assets is available when and where it is needed. This information includes details of how the assets have been configured and the relationships between assets. SACM supports the business by providing the information needed to manage all CIs across the whole of the service life cycle. This contributes to the success of all service management processes, as well as providing IT management and the business with the information needed to get maximum value from service assets. The scope of SACM may extend to non-IT assets and to internal and external service providers, where shared assets need to be controlled. To manage large and complex IT services and infrastructures, SACM requires the use of a supporting system known as the CM system. The main subprocesses of this phase are configuration identification, configuration control, and configuration verification and audit. More information about ITIL can be obtained from the ITIL home at http://www.itil-officialsite.com/.

Control Objectives for Information and Related Technology (COBIT)

COBIT is a framework developed by the Information Systems Audit and Control Association (ISACA) for IT management and IT governance. COBIT (currently version 5) is the only business framework for the governance and management of enterprise IT. COBIT 5 builds and expands on COBIT 4.1 by integrating other major frameworks, standards, and resources, including ISACA's Val IT and Risk IT, ITIL and related standards from ISO. Some of the benefits of using COBIT are the following:

- Availability of high-quality information to support business decisions;
- Achievement of strategic goals and realization of business benefits through the effective and innovative use of IT;
- Achievement of operational excellence through reliable, efficient application of technology;
- Maintenance of IT-related risk at an acceptable level;
- Optimization of the cost of IT services and technology.

COBIT contains 34 high-level IT processes described in the COBIT model that provide excellent guidance in establishing IT controls including those for SCM. The key management jobs for CM in COBIT are the following:

- To establish and maintain a configuration model;
- To define CIs, attributes, business models, and relationships;
- To approve requests for new CIs and changes to existing ones;
- To register new CIs;
- To establish and maintain a configuration repository and baseline;
- To approve changes to CIs and the CM system;
- To produce status and configuration reports;
- To verify and review integrity of the configuration repository;
- To conduct configuration audits and report nonconformances.

These activities are each assigned to a person who is responsible, accountable, consulted, and informed to ensure that the practices are not bypassed. For example, establishing and maintaining a configuration model is the responsibility of the configuration manager and IT administration head, while the CIO, head architect, business process owners, and auditors are consulted on that. The head of IT operations is accountable for ensuring that the configuration model is established and maintained. The model provides the roles and responsibilities of the key SCM personnel such as the configuration manager, configuration analyst, configuration administrator, CCB, CIO, head of IT operations, head of IT administration, head architect, head of development, and business process owners.

COBIT provides extensive guidelines and advice on how to implement the various CM activities, how to set up a CM system (CMS), and how to assign the various CM tasks, how to create a CM database (CMDB) and a host of other best practices to improve the efficiency and effectiveness of the CM system. More information about COBIT is available at ISACA's Web site, http://www.isaca.org/COBIT/.

Software Engineering Body of Knowledge (SWEBOK)

SWEBOK, a guide that establishes software engineering as a recognized engineering discipline, is aimed at promoting a consistent view of software engineering worldwide. The guide (SWEBOK) is not the body of knowledge, as the body of knowledge exists in the published literature. The purpose of the guide is to describe the portion of the body of knowledge that is generally accepted, to organize that portion, and to provide topical access to it. The body of knowledge is subdivided into 15 knowledge areas (KAs), which can be broadly categorized into three. The KAs are listed as follows:

- Development processes:
  - Software requirements;
  - Software design;
  - Software construction;
  - Software testing;
  - Software maintenance.
- Supporting processes:
  - SCM;
  - Software engineering management;

  - Software engineering process;
  - Software engineering models and methods;
  - Software quality;
  - Software engineering professional practice.
- Engineering fundamentals:
  - Software engineering economics;
  - Computing foundations;
  - Mathematical foundations;
  - Engineering foundations.

SCM is a supporting-software life cycle process that benefits project management, development and maintenance activities, and QA activities, as well as the customers and users of the end product.

The SCM knowledge area is further subdivided into seven subareas, described as follows:

- *Management of the SCM processes:* This subarea deals with topics like organizational context for SCM, constraints and guidance for SCM, planning for SCM, SCM plans, and surveillance of SCM.
- *Software configuration identification:* Software configuration identification identifies items to be controlled, establishes identification schemes for the items and their versions, and establishes the tools and techniques to be used in acquiring and managing controlled items. The topics in this subarea are identifying items to be controlled and the software library.
- *Software configuration control:* Software configuration control is concerned with managing changes during the software life cycle. The topics in this subarea are requesting, evaluating, and approving software changes and implementing software changes and deviation and waivers.
- *Software configuration status accounting:* Software configuration status accounting is an element of CM consisting of the recording and reporting of information needed to manage a configuration effectively. The main topics in this subarea are software configuration status information and software configuration status reporting.
- *Software configuration auditing:* This consists of software functional configuration auditing, software physical configuration audits, and in-process audits of a software baseline.
- *Software release management and delivery:* The main topics of this subarea are software building and software release management.
- *Software CM tools:* This subarea discusses the SCM tools. SCM tools can be divided into three classes in terms of the scope at which they provide support: individual support, project-related support, and company-wide process support.

The latest version of the SWEBOK is available free of cost in HTML format (http://www.computer.org/portal/web/swebok/htmlformat) and in PDF format (http://www.computer.org/portal/web/swebok).

Summary

CM is a vital discipline for all of the process improvement efforts and models that exist today. Process improvement models and frameworks like CMM, CMMI, SPICE, BOOTSTRAP, Trillium, ITIL, COBIT, and SWEBOK require a CM. To achieve certification, the existence of formal CM and procedures is a must.

Selected Bibliography

Bell Canada, Trillium: Model for Telecom Product Development & Support Process Capability (Release 3), Bell Canada, 1994.

Bourque, P., and R. E. Fairley (eds.), *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

Caputo, K., *CMM Implementation Guide*, Reading, MA: Addison-Wesley, 1998.

Chrissis, B. M., M. Konrad, and S. Shrum, *Introduction to CMMI*, Reading, MA: Addison Wesley, 2003.

CMMI Product Team, Capability Maturity Model® Integration (CMMI[SM]), Version 1.1: CMMI[SM] for Systems Engineering and Software Engineering (CMMI-SE/SW, V1.1), Continuous Representation, Technical Report (CMU/SEI-2002-TR-001), Software Engineering Institute, Carnegie Mellon University, 2001.

CMMI Product Team, Capability Maturity Model® Integration (CMMI[SM]), Version 1.1: CMMI[SM] for Systems Engineering and Software Engineering (CMMI-SE/SW, V1.1), Staged Representation, Technical Report (CMU/SEI-2002-TR-002), Software Engineering Institute, Carnegie Mellon University, 2001.

CMMI Product Team, Capability Maturity Model® Integration (CMMI[SM]), Version 1.1: CMMI[SM] for Systems Engineering, Software Engineering, and Integrated Product and Process Development (CMMI-SE/SW/IPPD, V1.1), Continuous Representation, Technical Report (CMU/SEI-2002-TR-003), Software Engineering Institute, Carnegie Mellon University, 2001.

CMMI Product Team, Capability Maturity Model® Integration (CMMI[SM]), Version 1.1: CMMI[SM] for Systems Engineering, Software Engineering, and Integrated Product and Process Development (CMMI-SE/SW/IPPD, V1.1), Staged Representation, Technical Report (CMU/SEI-2002-TR-004), Software Engineering Institute, Carnegie Mellon University, 2001.

CMMI Product Team, Capability Maturity Model® Integration (CMMI[SM]), Version 1.1: CMMI[SM] for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1), Continuous Representation, Technical Report (CMU/SEI-2002-TR-011), Software Engineering Institute, Carnegie Mellon University, Software Engineering Institute, Carnegie Mellon University, 2002.

CMMI Product Team, Capability Maturity Model® Integration (CMMI[SM]), Version 1.1: CMMI[SM] for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1), Staged Representation, Technical Report (CMU/SEI-2002-TR-012), Software Engineering Institute, Carnegie Mellon University, 2002.

CMMI Product Team, Capability Maturity Model® Integration (CMMI[SM]), Version 1.1: CMMI[SM] for Software Engineering (CMMI-SW, V1.1), Continuous Representation, Technical Report (CMU/SEI-2002-TR-028), Software Engineering Institute, Carnegie Mellon University, 2002.

CMMI Product Team, Capability Maturity Model® Integration (CMMI[SM]), Version 1.1: CMMI[SM] for Software Engineering (CMMI-SW, V1.1), Staged Representation, Technical

Report (CMU/SEI-2002-TR-029), Software Engineering Institute, Carnegie Mellon University, 2002.

Garcia, S. M., Evolving Improvement Paradigms: Capability Maturity Models & ISO/IEC 15504 (PDTR), Software Engineering Institute, Carnegie Mellon University, 1996.

Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

Paulk, M. C., A Comparison of ISO 9001 and the Capability Maturity Model for Software, Technical Report (CMU/SEI-94-TR-12), Software Engineering Institute, Carnegie Mellon University, 1994.

Paulk, M. C., et al., Capability Maturity Model[SM] for Software, Version 1.1, Technical Report (CMU/SEI-93-TR-024), Software Engineering Institute, Carnegie Mellon University, 1993.

Paulk, M. C., et al., Key Practices of the Capability Maturity Model[SM], Version 1.1, Technical Report (CMU/SEI-93-TR-025), Software Engineering Institute, Carnegie Mellon University, 1993.

Pressman, R. S., *Software Engineering: A Practitioner's Approach*, New York: McGraw-Hill, 2001.

SEI, ISO/IEC 15504: Frequently Asked Questions, Software Engineering Institute, Carnegie Mellon University, 1998.

Sommerville, I., *Software Engineering*, Reading, MA: Addison-Wesley Publishing Company, 2001.

Trillium Model Home Page (http://www.sqi.griffith.edu.au/trillium/).

Weber, C. V., et al., Key Practices of the Capability Maturity Model, Technical Report (CMU/SEI-91-TR-25), Software Engineering Institute, Carnegie Mellon University, 1991.

Zahran, S., *Software Process Improvement: Practical Guidelines for Business Success*, Harlow, England: Addison-Wesley, 1998.

CHAPTER 13

SCM Plans (SCMPs)

Introduction

We have seen that once the SCM system is designed, it should be documented so that the working of the SCM system, the procedures, and the functions, duties, and responsibilities of each member are transparent and known to all members of the SCM team, project team, the subcontractor's team (if any), and others. This document is called the SCMP or simply the plan.

An initial draft of the SCMP should be created and circulated among the various groups involved in the project during the initial phases (i.e., during the analysis or design phase). Once feedback from the various groups—project team, QA team, management, and others—is obtained, it can be incorporated, and the approved SCMP can be made available so that everybody is clear about the various SCM procedures and their duties and responsibilities.

According to Bounds and Dart [1], the CM plan is one of the three keys to the success of attaining a CM solution (the other two are the CM system and the CM adoption strategy). It is generally the case that an SCM solution is part of a corporate-wide process improvement plan and, as such, the solution is coordinated with that effort. This means that the SCMP needs to be in agreement with any other plans related to the corporate improvement effort.

An SCMP clearly describes how SCM is accomplished and how consistency between a system's configuration and the configuration records is achieved and maintained. The SCMP is a central source of information for the SCM program. The major benefits of creating a CM plan include an assurance that the appropriate SCM processes are applied, the responsibilities for the various SCM activities are assigned, detailed descriptions of these responsibilities are documented, accurate knowledge concerning required resources are available, and a foundation for continued improvement is put in place.

The objective of the SCMP is to create and document a system that will describe and specify, as accurately as possible, all tasks that are to be performed by the agency that is responsible for the configuration of the system or product. Thus the main function of the SCMP is to create an awareness among the various groups involved in a software project about the SCM functions and how they are to be performed in that project.

SCM is not a well-known subject and in most cases people—even people who have been in the software profession for many years—are not aware of SCM and how an SCM system works. So in order to create awareness among the project team members, SCM team members, and other people who are in some way related to

the project (such as the QA personnel), it is necessary to create a document that describes how SCM will be practiced for the project. Thus the SCMP forms the basis of training the personnel who are part of the project team and SCM team. It will be used as a reference manual for the SCM functions. It will also be used in the resolution of conflicts regarding the practice or implementation of SCM functions in the project.

The SCMP can be either created for the organization or for each project. If the plan is created for the entire organization, its suitability should still be assessed for each project, and necessary modifications should be made. In the case of organizations where the projects that are carried out differ in nature, complexity, and size, it is desirable to create separate plans for each project tailored to suit the needs and characteristics of the project.

SCMP and the Incremental Approach

Some companies adopt what is called the incremental approach to SCM implementation. In the incremental approach, SCM functions are implemented in stages. The system starts with just some of the components, say, a change- or problem-tracking system or a source code control and revision management system. Then slowly, as time progresses, the other components are introduced until the full spectrum of SCM functionality is achieved.

The natural question when considering the incremental approach is how an SCMP can be developed when the full SCM system is not being implemented. Another question is why an SCMP is needed since many of the components will not be part of the plan during the first phase of implementation.

In the author's opinion, irrespective of whether the company chooses the big-bang approach (where full SCM functionality is implemented in one shot) or the incremental approach, it is always better to have an SCMP. This is because, even for the incremental approach, the broad outline of what the final system will be like must be decided at the outset. There is no need to go into the finer details of the portions that are not being implemented in the first phase, but a very high-level outline of how these missing components will fit into the final system and the chosen implementation strategy (the when and how of implementing those functions) should be documented. This is important because, if the different functions of the SCM system are implemented without considering the overall picture (that is, the effect of each subsystem on the overall SCM system), then the integration of the different subsystems will not be seamless.

So, even when the incremental approach is used, the full SCM system should be designed and the SCMP prepared based on a full implementation. Otherwise, as components are added and full SCM functionality finally is reached, the SCM tools will not be well-integrated.

SCMPs and SCM Tools

Another question that is often asked about SCMPs by companies using SCM tools is whether the SCMP is really necessary. Here also the answer is, "yes." The SCMP

is the document that records how the different SCM functions will be performed. Only after analyzing and deciding how the SCM system should function, what its functions will be and which functions will be automated, and the peculiarities of each organization or project can the company decide which tool to use.

The SCMP has a section on SCM resources in which details are given about the software tools, techniques, equipment, personnel, and training necessary for the implementation of the specified SCM activities. The SCMP, as discussed earlier, is the basis for SCM training and auditing of the SCM system. So it is still very important to have an SCMP. In fact, the author strongly advocates training employees in the fundamentals and concepts of SCM and in how the SCM functions are carried out before training them in the tools. This will give the users a better understanding of what they are doing and how their actions will affect others in the project. In practice, if one is using a tool he or she can perform the SCM functions without knowing the SCM concepts; the tool's user manual is enough. If, however, users want to know why they are performing a function, they need to be aware of the SCM concepts.

So even though it is possible to use SCM tools without knowing much about the SCM concepts and functions, a person who knows the SCM concepts is a better user of the SCM tool than a person who has been told just to perform certain activities. There is always a difference between doing something just because somebody is told to do it that way and doing something after knowing why a task has to be performed and what will be the effect of that action.

SCMPs and Standards

Almost all CM standards advocate some sort of a document or plan. Except for some minor differences, the format specified by most of the standards for the SCMP is similar. According to a study conducted by Bounds and Dart [1], in which they compared three standards (by IEEE, NASA, and DOD) based on six criteria (ease of use, completeness, tailorability, consistency, correctness, and life cycle connection), the IEEE standard—IEEE Std-828-1998—had the best rating. IEEE Std-828-1998 defined only the contents of an SCMP. It has since been revised. The revised standard—IEEE Std-828-2012—expands the previous version to explain CM, including identifying and acquiring CIs, controlling changes, and reporting the status of CIs, as well as software builds and release engineering. It also addresses what CM activities are to be done, when they are to happen in the life cycle, and what planning and resources are required. In the revised version there is only an annex (Annex D) that deals with the CM plan (CMP) and what it should contain.

According to IEEE Std-828-2012 the CMP should have the following sections:

- Introduction: This section should cover such topics as the purpose, scope, relationship to the organization and other projects, key terms, and references;
- Criteria for identification of the CIs to which CM will be applied;
- Limitations and assumptions affecting the plan;
- CM responsibilities and authorities;
- Project organization;

- CM responsibilities;
- Applicable policies, directives, and procedures;
- Planned activities, schedules, and resources;
- CMP maintenance.

For developing a CMP, the combination of IEEE Std-828–1998 and IEEE Std-1042–1987 is still a better option than depending on the annex of the IEEE Std-828-2012.

As mentioned before, many standards exist for SCMPs. In the following sections, we will examine four, comprising a representative sample of the lot:

- IEEE Std-828–1998, IEEE Standard for Software Configuration Management Plans, and IEEE Std-1042–1987, IEEE Guide to Software Configuration Management;
- MIL-HDBK-61A(SE)-2001, Military Handbook: Configuration Management Guidance;
- EIA-649-B: 2011, National Consensus Standard for Configuration Management;
- ISO 10007: 2003, Quality Management-Guidelines for Configuration Management.

ANSI/IEEE Std-828–1998 and ANSI/IEEE Std-1042–1987

ANSI/IEEE Std-828–1998 is the standard for CMPs. Complementing this standard is a guide—IEEE Guide to Software Configuration Management (ANSI/IEEE Std-1042–1987)—that contains an explanation of the standard and sample SCMPs for different kinds of projects. These two standards together form the most comprehensive standards available on SCM and SCMPs.

The IEEE standard is a very comprehensive standard on SCM, and it can be customized easily to suit one's needs. This standard intentionally addresses all levels of expertise, the entire life cycle, other organizations, and the relationships to hardware and other activities on a project.

The ANSI/IEEE Std-828–1998 provides a format for creating the SCMP. According to the standard, the SCMP should consist of six sections as shown in Table 13.1.

ANSI/IEEE Std-1042–1987, the guide to SCMPs, explains how Std-828 should be implemented. This guide has several appendixes for different types of SCMPs (e.g., real-time, critical projects, and maintenance projects) that can be customized very easily to suit individual needs and requirements.

MIL-HDBK-61A (SE)-2001

The purpose of MIL-HDBK-61A (SE) is to assist configuration managers in planning for and implementing effective DOD configuration management activities and practices during all life cycle phases of defense systems and CIs. It supports acquisition based on performance specifications and the use of industry standards and methods to the greatest practicable extent.

Table 13.1 Format of SCMP per IEEE Std-828–1998

No.	*Section Name*	*Description*
1	Introduction	Purpose of the plan, scope, definition of key terms, and references.
2	SCM Management	Describes the allocation of responsibilities and authorities for SCM activities to organizations and individuals within the project structure.
3	SCM Activities	Identifies all functions and tasks required for managing the configuration of the software system as specified in the scope of the SCMP. Both technical and managerial SCM activities must be identified.
4	SCM Schedules	Establishes the sequence and coordination for the identified SCM activities and for all events affecting the SCMP's implementation.
5	SCM Resources	Identifies the software tools, techniques, equipment, personnel, and training necessary for the implementation of the specified SCM activities.
6	SCMP Maintenance	Identifies the activities and responsibilities necessary to ensure continued SCM planning during the life cycle of the project.

According to MIL-HDBK-61A (SE), CM planning is a vital part of the preparation for the next phases of a program life cycle. The CMP documents the results of that planning to enable it to be communicated and used as a basis in managing the program CM activities.

Appendix A of this handbook provides guidance in the content, use, and maintenance of government CMPs (GCMPs). It also provides guidance in evaluating contractor CMPs (CCMPs). In addition, the appendix provides two activity guides that give the contents and guidance for writing GCMPs and CCMPs.

The primary objective of the GCMP is to document the planning for the government CM activity to take place during the upcoming phase and to schedule specific actions necessary to implement those activities. The second purpose is to communicate and coordinate the government's intentions with the contractor or contractors involved in the program so that efficient and effective interfacing processes and working relationships may be established. The GCMP communicates to the contractor, the government's CM objectives for a given phase and the associated risks if those objectives are not met. It describes the expected deployment and use of the system or CI and indicates the CM process, systems, and methodologies the government plans to use and the interfaces that the contractor will be expected to establish. According to the handbook, the content of the GCMP should include the following six sections: (1) introduction, (2) reference documents, (3) government CM concept of operations and acquisition strategy, (4) CM organization, (5) data management, and (6) government CM process.

The CCMP describes the contractor's CM objectives, the value-adding CM activities that will be employed to achieve them, and the means of measuring and assuring that they are effectively accomplished. The CCMP should have the following sections: (1) introduction, (2) reference documents, (3) CM organization, (4) CM phasing and milestones, (5) data management, (6) configuration identification, (7) interface management, (8) configuration control, (9) CSA, (10) CAs, and (11)

subordinate performing activity and vendor control. The handbook gives detailed descriptions and phase-by-phase guidance for each section.

EIA-649-B: 2011

According to EIA-649-B, the purpose and benefits of CM planning and management include the following:

- Assurance that the appropriate CM processes and activities are applied;
- Establishment of organizational responsibilities for CM activities;
- Identification of—and making available—necessary resources and facilities;
- Formation of a basis for continuous improvement;
- Enhanced maturity of the enterprise's process.

EIA-649-B discusses in detail the requirements of the plan and considerations that should be taken into account in its development. A CMP should not be a one-size-fits-all document. Rather, it must be tailored to meet the needs of the agency responsible for CM. In particular, EIA-649-B points out that a well-developed plan will aid in the training of personnel in CM and will help explain the process to outside personnel, such as upper-level management and auditors. A comprehensive CMP that reflects efficient application of CM principles and practices to the identified context and environment would normally include the following topics:

- General product definition and scope;
- Description of CM activities and procedures for each major CM function—configuration planning and management, configuration identification, configuration change management, CSA, configuration verification and audit, and CM of digital data;
- CM;
- Organization, roles, responsibilities, and resources;
- Definitions of terms;
- Programmatic and organizational interfaces;
- Deliverables, milestones, and schedules;
- Subcontract flow-down clauses;
- Definitions of terms.

If these topics are covered in detail in the plan, the CM program will have a sound blueprint to guide its effective implementation.

ISO 10007: 2003

This international standard gives guidance on the use of CM in industry and its interface with other management systems and procedures. This standard defines a CMP as a document that sets out the organization and procedures for the CM of a specific product or project. The standard states that the CM plan provides the CM procedures that are to be used for each project and states who will undertake these

and when. The standard also states that the plan should be subjected to document control procedures.

This standard specifies a format for the CM plan (Annex A-Recommended Structure and Content of a Configuration Management Plan). According to ISO 10007, a CMP should have the six chapters listed in Table 13.2.

The standard also specifies that whenever an existing procedure or standard is used, it should be referenced rather than repeated so that duplication can be avoided and simplicity can be maintained. This is a good practice to follow irrespective of which standard you are using, because it will make the SCMP simple and short, and it will eliminate data duplication. For example, according to the preceding standard, the audit procedures that will be followed have to be mentioned in Chapter 6, CA. If the audit is conducted as per the ISO guidelines for auditing quality systems, instead of defining and describing the auditing process, the plan can just say that the SCM auditing will be done in accordance with the ISO 10011–1:1990, ISO 10011–2:1990, and ISO 10011–3:1990 (ISO guidelines for auditing quality systems, Parts 1, 2, and 3).

In a study conducted by Bounds and Dart [1], users of the CMPs were asked whether CM procedures should be part of the CMP or separate. The overwhelming response was that the procedures should be kept separate from the plan but that the plan should reference the procedures. Although many reasons were cited for this, the most common reasons were that separating the procedures allows the users to focus only on what applies to them and makes maintenance of the procedures and plan much easier. Respondents also stated that procedures should focus on how to do something, whereas a plan should focus on what is to be done.

The same study recommended the use of IEEE standards as the best standards to use in developing SCMPs for these reasons:

Table 13.2 Format of SCMP per ISO 10007

No.	*Chapter Name*	*Description*
1	Introduction	Description of the system or CIs to which the plan applies, a schedule of the CM activities, the purpose and scope of the plan, list of related documents, and so on.
2	Policies and Procedures	CM policies, CM organization and structure of the CCB and the other committees, selection criteria for the CIs, frequency, distribution, and control of reports and agreed terminology.
3	Configuration Identification	Family tree of the CIs, numbering conventions, baselines to be established, and so on.
4	Configuration Control	Organization and composition of the CCB, change management procedures, and so on.
5	CSA	Procedures for collecting, recording, processing, and maintaining the data for status accounting reports, definition of all CM reports, and so on.
6	CA	List of audits to be conducted, the audit procedures, the authorities and disciplines involved, format of the audit reports, and so on.

- The IEEE standard was written explicitly for use by anyone within the industry, whereas the NASA and DOD standards were written for their specific segments of industry.
- The IEEE standard is, by far, more complete than the other two standards and is the only standard that can be treated as a stand-alone document.
- The IEEE standard has greater potential for timely updates than the other standards since it is used by general industry.

Audit of the SCMP

An SCMP is a controlled document as well as a CI. So all document control procedures that are applicable to a controlled document and all the change management procedures that are applicable to a CI are applicable to the SCMP also. This means that the distribution of the SCMP should be controlled. There should be a distribution list that contains the names of persons having a copy of the plan. Also, access to the plan should be controlled. The level of control is decided based on the nature of the project. If the plan is hosted on the company intranet or bulletin board, then access to that should be controlled.

Another aspect is that changes to the plan should be done in accordance with the change management procedures mentioned in the plan. When a change is implemented the new versions should be made available to all who are in the distribution list. If the plan is hosted on the intranet then it should be updated.

The SCMP should be subject to auditing. Like any other CI, the plan should undergo the functional and physical CAs. Auditing ensures that the plan is complete and correct and satisfies the requirements as described in the standard or standards on which it is based.

How to Write a Good SCMP

Writing the SCMP is not an easy task. Good SCMPs take time; thorough knowledge of SCM functions, the peculiarities of the project, and the organization; and knowledge of other procedures such as auditing and testing. It is also not a task that should be taken lightly because the practice of SCM functions in a project or organization is based on the procedures and tasks specified in the SCMP. It is not too strong a statement to say that the SCMP can make or break a project. A bad and improperly designed SCMP will create unnecessary and cumbersome procedures, and instead of assisting the development process and improving productivity, it will result in confusion and an increased workload for the project team and the SCM team. In the following paragraphs, we look at some practices and tips that will help in the creation of good SCMPs.

The most important decision that affects the quality of an SCMP is the capability and knowledge of the person or the team that writes the plan. Ideally, the plan should be written by the people who have designed the SCM system for the project or organization. Here we are talking about experienced people who have a good understanding of the project or organization and the SCM system. Once the

plan is written, the technical documentation team can copyedit it so that typos and grammatical mistakes are eliminated. This is a good practice to follow, because the writing skills of the technical people may not be on par with their technical skills. The copyeditors should be given clear instructions not to touch the structure of the plan, just to check for grammatical and spelling errors. In fact, it could be beneficial to have the editor sit with the technical team during the writing process, so if any issues arise, they can be resolved immediately.

The second most important decision is selecting the standard on which the SCMP is going to be based. As mentioned earlier, the IEEE standards emerge as the best and most popular choice. However, it is good practice to see what other standards have to offer, and it may not be a bad idea to borrow good ideas from them. There is nothing wrong with formulating an SCMP based on more than one standard, because all standards will have some weak areas, and adapting substitute areas from other standards might improve them. Some of the standards that could be referenced are listed as follows:

- IEEE Std-828–1998, IEEE Standard for Software Configuration Management Plans, IEEE, 1998;
- IEEE Std-1042–1987, IEEE Guide to Software Configuration Management, IEEE, 1987;
- ISO 10007, Quality Management-Guidelines for Configuration Management, ISO, 2003;
- DOD MIL-STD-973, Military Standard for Configuration Management, Department of Defense, 1995;
- MIL-HDBK-61A (SE)—Military Handbook: Configuration Management Guidance, 2001.

One can get a feel for how to write the SCMP by studying some sample plans and reading books on SCM. Today, getting sample SCMPs is not a difficult task. IEEE Std-1042–1987 has four appendixes consisting of sample SCMPs for different types of projects. Also, hundreds of SCMPs—of all types and sizes—are available on the Web. A Google query for "sample SCMP" produced more than 2,050,000 results.

The next step in writing the SCMP is to identify the procedures that should be followed in the practice of SCM. If the SCM system is using procedures that are part of other standards, then as we have seen before, it is quite sufficient just to give references to those standards. These standards can be industry standards, the organization's internal standards, or even the project's own standards. However, before giving reference to a standard, it is a good idea to ensure that all the standards are available and, if possible, to bring them under document control, so that they are readily available for reference.

We have researched the standards and literature, discussed the sample plans, and identified the procedures that will be followed and the documents that define these procedures. The next step is to write the plan using any of the existing templates that are available. Many SCMP templates are available on the Web; the IEEE standard provides a very customizable template, and the ISO has a reasonably good template. The choice of template is a matter of taste and convenience. The contents of both the IEEE and ISO templates are almost the same, but the IEEE template is more comprehensive. For a company using ISO standards, however, the ISO

10007 template can be used. You just have to choose a template that is suited to your purpose (and similar to your project or organization) and customize it to your specific needs. Once the template or the table of contents is ready, the next step is to fill in the blanks or put the procedures and other details in place to complete the plan. The resulting document is considered the initial draft of the SCMP. Copies of this document—the initial draft—should be circulated to all groups that will be involved in implementing the SCM system and performing the various SCM functions. This process—involving everyone who matters in plan development—is very important; SCM is a team effort. To implement and manage an SCM system successfully, the SCM team will need cooperation from all quarters. One way to ensure that cooperation is to get others involved by sending copies of the initial draft and asking for feedback.

Once the feedback from the various groups is received, the SCMP's authors can review it, accept valid comments, and incorporate them into the plan. Once the final draft is ready, it is a good idea to get it reviewed by an external agency—a person or team of experts. This review can throw light on issues that the plan might have failed to address or bring up inadequacies or even detect errors that the internal reviews have missed. The external audit also provides a stamp of approval from a body that is supposed to be an expert in this area, which will help to increase the credibility and acceptance of the plan. Once the SCMP is reviewed and approved, it can be baselined.

Contents of a Typical SCMP

The SCMP can be written in any format as long as it contains all necessary information. The standards offer considerable latitude and freedom to the person who writes the SCMP. All standards require one to address certain topics such as scope, purpose, definitions, SCM organization, SCM functions, responsibilities, and resources. How this material should be presented, however, is decided by the author of the plan. Such decisions as the degree of detail or the amount of additional information to include in the SCMP depend on the nature of the project.

A sample outline for an SCMP is given next. All items, sections, and subsections need not be present in all projects. Some will have additional information. This structure relies heavily on the IEEE standards. Each section and subsection is proceeded by an explanation of its contents.

i. Cover Page

- This page should have the title "SCMP" and details on the project such as the organization, the authorities, the version number, and the release date.

ii. Copyright Page

- This page should list the copyright information of the SCMP.

iii. Distribution List

- This page should include the name and number of copies distributed and a description of how the documentation control activities will apply to this document.

iv. About the Document

- A short description of the document and its sections.

1.0 INTRODUCTION

- An overview of the plan, the SCM activities, the audience for the plan, and how to use the plan, so that the user will have a clearer understanding of the plan. The introduction should contain at least the following four topics: purpose, scope, definitions, and references.

1.1 Purpose

- Addresses the need for the plan and the intended audience.

1.2 Scope

- Covers the plan's applicability, limitations, and assumptions. This section provides an overview of the software development process in the project or organization and how the SCM functions and activities fit into the project.

1.3 Definitions

- Defines the key terms used in the document.

1.4 References

- Identifies all documents, standards, and external and internal procedures to be used in the plan. This section also identifies where the documents can be found so that the readers of the plan can retrieve them.

2.0 SCM MANAGEMENT

- Gives information on the organization of the SCM team and the allocation of responsibilities to teams and individuals, among other management topics.

2.1 SCM Organization

- Describes the organizational structure of the SCM team and how it fits into the organizational structure with respect to other groups such as the project team, the QA team, and top management. Also included in the structure are clients (customers and vendors) and subcontractors, if any are involved in the SCM activities. An organization chart depicting the structure is very useful in this section. This section also describes the composition of the CCB and other auditing and review teams that will be part of the SCM activities.

2.2 SCM Responsibilities

- Describes the duties and responsibilities of all those involved in carrying out the SCM activities. This section identifies the responsibilities of the CCB and other committees and boards necessary for CM, the structure of which is defined in the previous section.

2.3 Relationship of SCM to the Software Process Life Cycle

- Relates the SCM activities to the different phases of the software development life cycle. It spells out what SCM activities need to be performed during each phase of the life cycle.

2.4 Interfaces to Other Organizations on the Project

- Describes how the SCM team will interact with other organizations in the project such as QA, test, project management, and requirements and including vendors and subcontractors.

2.5 SCM Responsibilities of the Organizations

- Describes the responsibilities—or what is expected—of the vendors, subcontractors, and other organizations in relation to the carrying out of SCM functions.

3.0 SCM ACTIVITIES

- Identifies the tasks and functions that are required to manage the configuration of the system as specified in the scope of the plan. This section deals with the core SCM activities and how they are performed in the project.

3.1 Configuration Identification

- Describes how to identify, name, and document the functional and physical characteristics of the CIs. Once the items are identified, they are acquired and moved into the controlled environment.

3.1.1 Identification of CIs

- Identifies the items to be selected as CIs that will be controlled by the SCM activities. This section gives a list of CIs in the project. Inclusion of a tree structure showing the various CIs and their interdependencies is ideal.

3.1.2 Naming CIs

- Specifies the identification system, naming conventions, version numbers, and letters used to identify the CIs.

3.1.3 Acquiring CIs

- Describes how the CIs are to be stored, how access to them will be controlled, the details of the configuration libraries, the procedures for check in and check out of CIs from the library, and other related information.

3.2 Configuration Control

- Explains the change management processes such as change initiation, change disposition, change implementation, reviews, approval, and baselining.

3.2.1 Change Initiation

- Describes how to initiate a change. A change can be the result of a fault or problem or the result of an enhancement or new feature. This section describes the procedures to be followed to initiate a CR or PR so that the change management activities are started.

3.2.2 Change Evaluation

- Describes how the evaluation of a CR is carried out, including details on handling problem analysis and problem classification. The section details how to classify the changes or problems and how to do an impact analysis, among other relevant information. The section also specifies the qualifications of the people who will be doing the change or problem evaluation.

3.2.3 Change Management

- Describes how a CR is processed. It spells out clearly procedures such as those for receiving CRs, assigning CRs for evaluation, CCB meetings, and dispositioning the CRs carried out.

3.2.4 Change Implementation

- Once the CR is approved it has to be implemented. Selecting the change implementation team or person, conducting verification and validation, and promoting the item to the new baseline are described in this section.

3.2.5 CCBs

- This is the apex body that decides the fate of the change requests. This section describes the functioning of the CCB. If multiple CCBs are present, the authority of each CCB must be specified and if more than one CCB of the same authority is present in the project, then conflict resolution mechanisms also should be documented.

3.3 CSA

- Details the recording the status of the CIs and reporting them to people who need to know about them.

3.3.1 Identification of Information Needs

- Describes the information requirements of the project, including what kind of information is required, who requires it, the nature of the requirement (e.g., routine or ad hoc), and the frequency of the reports.

3.3.2 Information Gathering Mechanisms

- Explains how status accounting information is gathered. Ideally the information should be entered into the CMDB by the initiators of the SCM activities rather than by the SCM person chasing the activities and updating the status accounting data. For example, when a CR is initiated, if the person who initiates the CR creates a record of that in the database, the job of information gathering is easy. However, to accomplish this, the necessary forms and access privileges should be given to the different users of the system. This section describes the exact mechanism of capturing the information for status reporting.

3.3.3 Reports, Their Contents, and Frequency

- Describes the various reports that will be created, their contents, and the frequency of each report.

3.3.4 Access to Status Accounting Data

- The status accounting function cannot anticipate all of the information requirements of users and produce reports to meet all requirements. Also, in many cases, information requests will be for ad hoc reports, which may be generated only once. If the status accounting system is computerized, then an interactive query facility can be made available to users to get this information. If such a facility is available, this section will describe the procedures for using that facility. In the case of manual processing, this section will describe how the manual records can be accessed for ad hoc information needs.

3.3.5 Status Accounting Information Dissemination Methods

- Describes how and to whom the status accounting information will be disseminated.

3.3.6 Release Details

- Details information such as what is contained in a release, to whom the release is being provided and when, the media the release is on, known problems with the release, known fixes in the release, and installation instructions.

3.4 Configuration Auditing

- Describes, for example, what types of audits are to be performed, the audit procedure, frequency, and the auditing authority.

3.4.1 Audits To Be Performed

- Describes the different types of audits that will be performed and when they will be performed. Typical audits include FCAs, PCAs, subcontractor audits, and external audits.

3.4.2 CIs Under Audit

- Specifies the list of CIs that are to be audited.

3.4.3 Audit Procedures

- Describes the procedure to be followed for each audit, including the auditing authority, the documents required, how the audit should be conducted, and the format of the audit report.

3.4.4 Audit Follow-Up Activities

- Describes the activities that should be carried out after the audit such as resolution of NCRs.

3.5 Interface Control

- Describes the coordination of the changes to the CIs with the changes to the interfacing items outside the scope of the plan like the hardware system, off-the-shelf packages, and support software. The plan must identify each of these external items and should define the nature of the interface, the affected groups, how the interface items will be controlled, and how the interface items will be approved to be included as part of a baseline.

3.6 Subcontractor or Vendor Control

- Describes the activities necessary to incorporate the items developed outside the project environment into the project environment, in particular, items that are the responsibility of subcontractors and vendors. This section should describe the SCM functions and activities that should be followed by the vendor or subcontractor, mechanisms to ensure that they are followed, procedures to audit the items that are submitted by the vendor or subcontractor, and the items that must be supplied by the vendor or subcontractor. This section also describes how the items will be received, tested, and placed under SCM and how CRs to these items will be processed and implemented.

4.0 SCM SCHEDULES

- Describes the sequence of the SCM activities, their interdependencies and relationship to the project life cycle, and project milestones. The schedule will identify the life cycle phases or project milestones where the different baselines (e.g., functional baseline, allocated baseline, and product baseline) will be established. This section also establishes the schedule for the different CAs. Graphical representation using PERT charts or Gantt charts help to enhance the usefulness of this section.

5.0 SCM RESOURCES

- Identifies the software tools, techniques, equipment, personnel, budget, and training necessary for the implementation of the specified SCM activities.

6.0 SCMP MAINTENANCE

- Describes the activities that are required to keep the plan current during the life cycle of the project. The plan should be monitored and synchronized with the activities of the project. This section describes the mechanism for synchronization and identifies the person or team responsible for those activities.

Sample SCMPS

We have just looked at a generic structure for SCMPs that can be tailored to suit the needs of individual projects. We have also discussed tips for writing good SCMPs. As mentioned previously, it is a good idea to go through a few sample SCMPs of similar projects before you start writing the SCMP.

Sample SCMPs can be obtained from a host of sources, but two of the easiest sources are listed as follows:

- *The Internet*. Thousands of SCMPs covering a spectrum of projects—e.g., military, government, research, and commercial—of various sizes and complexity are hosted on the Internet. You can use a search engine to locate them. By spending a few hours on the Internet, you can browse through the different types of SCMPs.
- *ANSI/IEEE Std-1042–1987*. This is the IEEE Guide to SCM. There are four appendixes for this document, which are sample SCMPs for different types of projects.

Also, anyone who is writing an SCMP will benefit tremendously from [1], which is an excellent primer for SCMPs.

Summary

The SCMP is the document that defines the SCM system and how SCM is to be practiced in a project. The SCMP documents what SCM activities are to be done, how they are to be done, who is responsible for doing specific activities, when they are to happen, and what resources are required.

The SCMP should be prepared irrespective of whether the organization is using an incremental approach or using SCM tools. Preparing a good SCMP requires knowledge of the SCM concepts and functions, SCM tools, SCM standards, software engineering and QA procedures, and standards and knowledge of the organization or project for which the standard is being written. The SCMP can be based on a single standard or can be based on more than one standard.

The SCMP is a CI and should be updated and reviewed whenever required. The SCMP should be prepared with the cooperation of all those who will be involved in the functioning of the SCM system and should be audited by external SCM experts. It is a good idea to go through some SCMPs before starting to write the plan. Hundreds of sample plans are available on the Internet. Also, anyone who is writing an SCMP will benefit tremendously from the document by Bounds and Dart [1].

Reference

[1] Bounds, N. M., and S. Dart, "CM Plans: The Beginning to Your CM Solution," Technical Report, Software Engineering Institute, Carnegie Mellon University, 1998.

Selected Bibliography

Arthur, J. D., et al., "Evaluating the Effectiveness of Independent Verification and Validation," *IEEE Computer*, Vol. 32, No. 10, October 1999, pp. 79–83.

Babich, W. A., Software Configuration Management: Coordination for Team Productivity, Boston, MA: Addison Wesley, 1986.

Ben-Menachem, M., *Software Configuration Management Guidebook*, New York: McGraw-Hill, 1994.

Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, 1992.

Bersoff, E. H., V. D. Henderson, and S. G. Siegel, *Software Configuration Management, An Investment in Product Integrity*, Englewood Cliffs, NJ: Prentice-Hall, 1980.

Bourque, P., and R. E. Fairley (eds.), *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition), Piscataway, NJ: IEEE, 2003.

IEEE, *IEEE Standard for Configuration Management in Systems and Software Engineering (IEEE Std-828–2012)*, Piscataway, NJ: IEEE, 2012.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

TechAmerica, *Configuration Management Standard (ANSI/EIA-649-B)*, Washington, DC: TechAmerica, 2011.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

CHAPTER 14

SCM Organization

Introduction

CM—like any other management function—needs people to perform the various activities and to produce results. CM is a discipline that applies technical and administrative direction and surveillance to (1) identify and document the functional and physical characteristics of a CI, (2) control changes to those characteristics, (3) record and report change processing and implementation status, and (4) verify compliance with specified requirements [1]. As you can see, this definition states a lot of tasks—configuration identification, change management, change disposition, change implementation, configuration control, status accounting, and CAs, among others—that have to be performed for the SCM system to function properly.

To perform all of these functions effectively and efficiently, one needs procedures and resources. In examining the various SCM procedures in the last few chapters, it becomes obvious that the most important resource is people. Accordingly, for any SCM system to function properly, there should be enough qualified people to do the various functions.

The number of people on an SCM team will vary depending on the nature, size, and complexity of the project. In the case of large and complex projects with hundreds of programmers and thousands of programs, the SCM team will be big with lots of full-time members, whereas in the case of small projects the project leader might do all the SCM functions independently. Also, there are people whose services will be required on an as-needed basis. For example, for a CR evaluation, an outside expert might be called, and once the evaluation is over and the report is submitted he or she will leave. There might also be permanent personnel on the SCM team who are in charge of receiving the various CRs and PRs, ensuring the completeness of these forms, assigning them to the right people for evaluation, coordinating the CCB activities, and whatever else is necessary. This chapter reviews the structure of a typical SCM team based on the different functions.

SCM and the Organization

We know that SCM is a support function, so let's take a look at where and how SCM fits into the organizational structure. Different organizational structures in different organizations will require the SCM team to be structured or positioned differently. In many cases, there will be a central SCM team that will take care of

all the SCM activities of the different projects in the company. The SCM team here will act as a support function and will use the members of the project team to get the SCM activities done.

The main responsibilities of the SCM team in such an arrangement are to complete the SCM activities of the different projects such as receive the CRs, assign the implementation, and convene the CCB meetings. The advantage of this kind of setup is that the central SCM team can have standardized procedures and policies enforced for all projects and prioritize the SCM needs of the different projects based on the overall company objectives. This kind of an arrangement is shown in Figure 14.1.

Some companies have a central SCM team along with individual teams for each project. The central team creates the guidelines and policies and the general organization-wide SCM plan and is responsible for the proper functioning of the SCM system in the company as a whole. The individual teams associated with each project customize the plans and procedures to suit the needs of the particular project and are responsible for the SCM activities in the project. These SCM teams in the different projects usually have a dual reporting arrangement in which they report to the central SCM team leader and to the leader of the project with which they are associated. This type of arrangement is shown in Figure 14.2.

In a third situation, there may be no central SCM team, but each project will have its own SCM team. This is usually applicable to large projects, where the size and complexity of the project warrants a full-fledged SCM team of its own. This type of arrangement is shown in Figure 14.3.

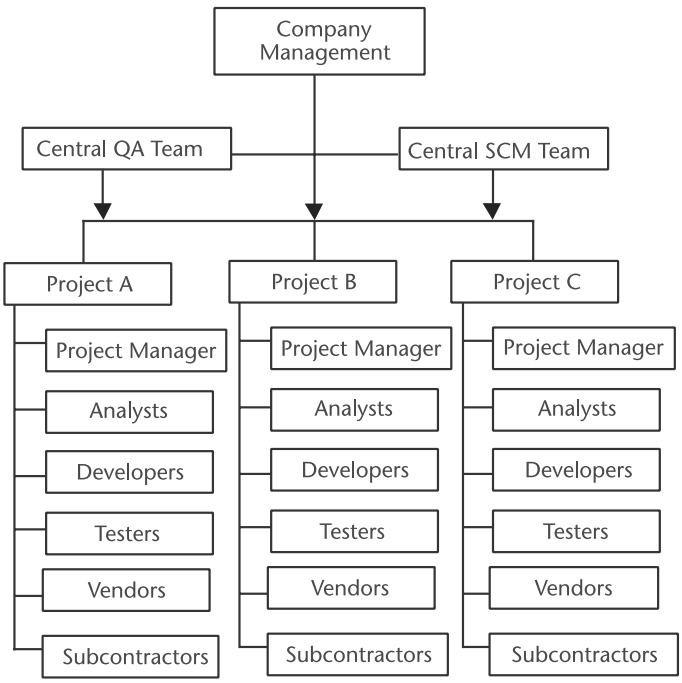


Figure 14.1 Organizational structure in which a central SCM team deals with the different projects.

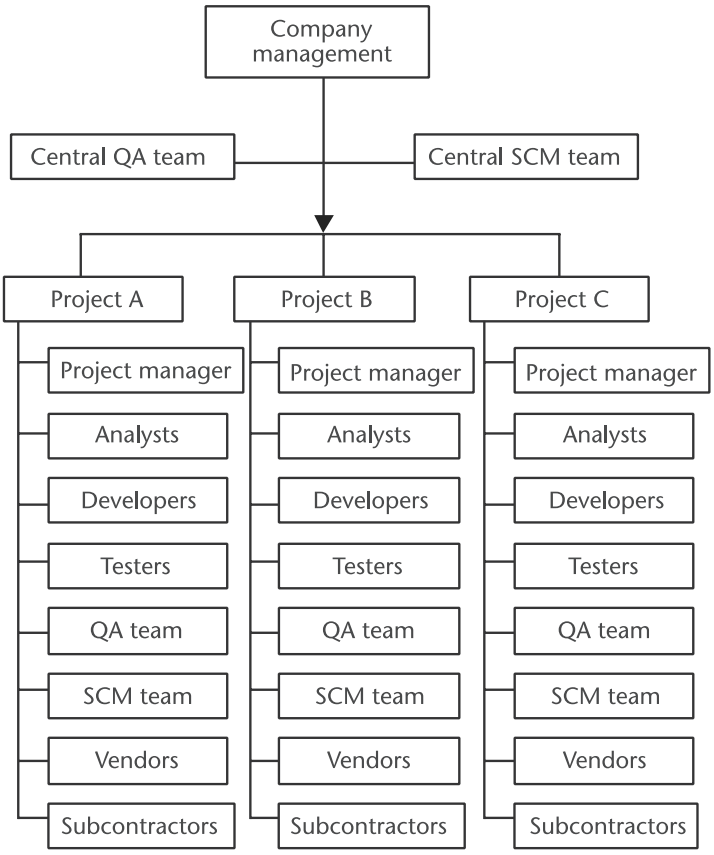


Figure 14.2 Organizational structure when a central SCM team and individual SCM teams are used.

The SCM team needs strong support from management, because it does not have the necessary muscle power to enforce its decisions like the line functions do. However, if proper awareness is created about the importance of SCM and its benefits and if the SCM system is designed in such a way that SCM approval is a prerequisite for moving from one phase to another, the SCM team can get the necessary cooperation from the project team and other support functions to carry out its tasks.

Another important aspect that should be remembered is that the SCM team will have to enlist the help of other professionals in the organization. The SCM team will require help from the management team, the QA team, the project team, and others to carry out the various functions. For example, the CCB's permanent members include management, QA, marketing, and project team representatives. Also, for conducting CR evaluations or causal analysis, outside help might be required. So a mechanism should be in place for determining how these human resources will be made available to the SCM team. Ideally, the SCM team should make a request to the concerned group and the group should cooperate. However, if the cooperation is not forthcoming, the SCM team might need management support to get the required personnel.

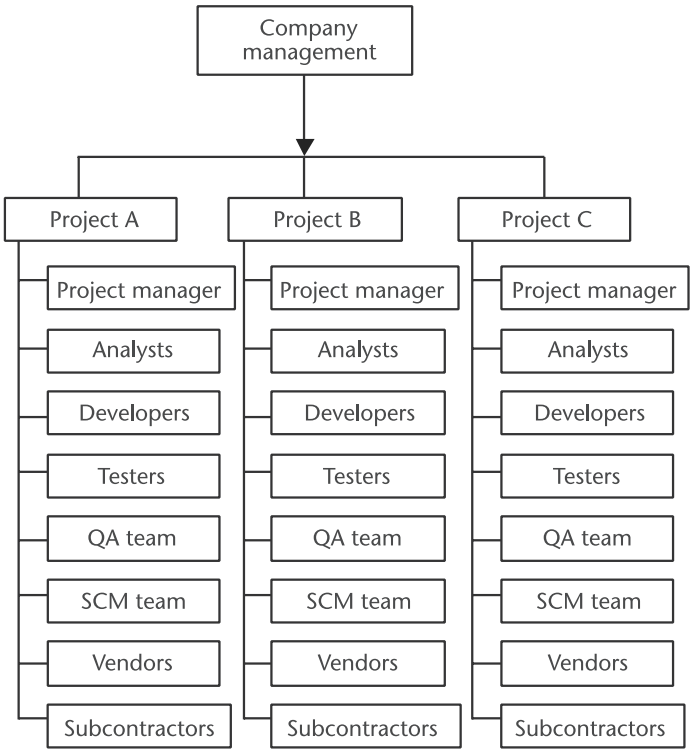


Figure 14.3 Organizational structure in which there are independent SCM teams for each project.

SCM Organization

The CMO is the head of the SCM team. He or she is usually the person who was part of the SCM system design team and has most probably written the SCM plan. This person is responsible for all SCM activities in the project and reports to the CCB. The main responsibilities of the CMO include setting up the SCM system, training the SCM team, assigning duties and responsibilities, constituting the CCB(s) as the case may be, coordinating between the project team and the CCB, managing the change control activities, setting up the SCM libraries and other control mechanisms, and coordinating between the audit team and the project team. He or she will be part of the CCB and will ensure that the CCB meetings are convened according to the schedule or whenever the need arises (emergency meetings).

Other members of the SCM team help the CMO in performing the SCM tasks. These include both technical people and administrative personnel. The technical people will be concerned with tasks like configuration identification, library management, change management, version control, release management, CAs, causal analysis, and updating the configuration knowledge base. The administrative staff will—among other things—ensure that meeting information is sent on time, the minutes of the meetings are sent to the people on the distribution list, the decisions made in the meetings are conveyed to the appropriate people, the status accounting reports are delivered, and the skill inventory database is updated regularly.

Then there will be people who work with the SCM team on an ad hoc basis. These people include those who conduct the CR evaluation, problem analysis, causal analysis, and CAs and those who serve as subject experts in the CCB meetings.

So in a typical SCM setup, different people will carry out the various SCM functions. Some of the people in an SCM environment are listed next and a brief description given about what they do.

- *Developers*. Developers are the project team members who develop the software system or product. They do the analysis, design, and coding.
- *Testers*. Testers conduct the testing of the programs developed by the developers. In most cases, the developers do the unit testing and hand over their programs to the testers. The testers do the module testing, the integration testing, and the system testing. These people are responsible for coordinating the alpha and beta testing phases. The testers originate the defect or problem reports, make enhancement suggestions, and collect and collate the feedback from the alpha and beta testing programs. Then they initiate the problem tracking and change management process for each of the defects or enhancements found. The role of the testers varies slightly from project to project (for example, in some projects, there will be separate teams for alpha and beta testing), but their main responsibility is to find the bugs and report them.
- *QA representatives*. The goal of a good QA program is to prevent defects from occurring or recurring. So the main responsibility of the QA personnel is to develop standards and guidelines for the different activities in the project such as design, coding, and testing. They must also make sure that these policies and standards are followed by everybody on the team. To ensure this, they conduct quality audits. QA personnel also form part of the CCB and play a vital role in the change disposition. In many organizations, it is the QA team members who perform the CR evaluation, problem analysis, casual analysis, and the knowledge base and help desk maintenance.
- *Assigners*. An assigner is the SCM team member responsible for scheduling the tasks that are to be performed based on their severity and impact and assigning these tasks to the other people in the team. Assigning the CRs for evaluation, giving the problem reports for analysis, giving the task of implementing a change to somebody, and ensuring that all activities are performed on schedule are all the assigner's responsibility. The assigner's function may also be performed by a pre-CCB screening committee [sometimes called the software review board (SRB)]. The SRB may also be given some measure of authority, by the CCB, to act on minor CRs and problem reports; such as, approvals, rejections, and requests for additional information.
- *Build manager*. This is another SCM role whose responsibility is to handle the various builds and releases. He or she is the person who is responsible for such tasks as ensuring that the CIs are given the correct version numbers, proper baselines are established, build files are accurately kept, and branching and merging of the file or files is done properly. The primary goal of this person is to take all the necessary steps to ensure that the SCM system is capable of configuring and building the system or its components completely and accurately at any time.

- *Administrator.* This is the person who does the database administration of the various SCM databases and repositories, assigns access privileges to the different team members, and makes backups, among other responsibilities. This person works very closely with the build manager to ensure that the build process proceeds without any problem.

We have already seen that many other people are involved in the SCM team. The roles just mentioned are generally full-time jobs and are usually present in every project that has SCM. The other personnel in the SCM team, possibly with the exception of the CMO and a few administrative people, are called in as needed. For example, the CCB members are called only in situations such as when a CR has been submitted that needs to be resolved or when a decision has to be made about the release of a product.

Automation and SCM Team Size

With CM tools automating every possible aspect of CM, the number of people required to manage the SCM functions is decreasing. However, some areas—analysis, evaluation, and audits—still require human intervention. Also, not all projects will use totally automated SCM tools; some projects will use tools that automate certain areas such as change management, version control, build management or system building, and status accounting. So depending on the nature of the project and level of automation, the organization of the SCM team will vary.

Another factor that should be considered is the additional capabilities, features, and functionality offered by the new-generation SCM tools. These tools are highly complex and sophisticated and hence require highly trained specialists like system administrators, database administrators, and release and build managers to manage them effectively and efficiently. So even though SCM tools automate the repetitive and monotonous activities, with the added capabilities, the number of people required to manage the SCM system has not decreased considerably. The advantages of automation include that people do not have to perform the repetitive and tedious tasks, that the chances of errors are less, and that accurate and up-to-date information is available, but not a significant reduction in the SCM team size.

Skill Inventory Database

Since the SCM team relies heavily on professionals from other groups in the organization to carry out its various functions., the SCM team should know whether the people (with the necessary qualifications) whom they want are available and if available where are they located and so on. The author has worked on many projects where one of the main problems faced by the SCM team was tracking down the right people to do particular functions, such as impact analysis, problem evaluation, and causal analysis.

In one particular project where the author was the CMO, this problem—the task of finding the right people—was very acute. The company had more than 1,500

employees in five or six different offices. The skills and availability of people was difficult information to get. So we used the idea of a skill inventory database. The concept was borrowed from industrial engineering, where creating a skill inventory of the shop floor workers has been used for implementing modern production systems like Toyota's. It is also used in small group manufacturing where finding the people with the right skills fast is a necessity.

Using a skill inventory database in the SCM system proved such a success that I have since used it in many projects with equal results.

Accordingly, it is a good idea for SCM teams to create a skill inventory database of the company's personnel. The database can store the details of each and every professional whose services will be required by the SCM team. This includes top management, the QA team members, the project team members, the SCM team members, the vendor and subcontractor team members, the members of the hardware group, and other support functions. An example of such a database is shown in Table 14.1.

A skill inventory database will come in handy in large organizations where the SCM manager or team needs to know details about staff, such as who can conduct an impact analysis, who has the necessary qualifications to be part of the CCBs, and who can conduct reviews, FCAs, and PCAs. In big organizations, the SCM team may not know all the employees and the skill set each person has. Employee turnover adds to this problem. CCB members who have left the organization have to be replaced, orphan CIs (CIs whose authors or owners have left the company or have been promoted and have more responsibilities than doing an impact analysis, for example) have to be replaced. So it is very helpful for the SCM team to have a skill inventory database to help find the right people for different tasks.

The skill inventory database captures the skills of every person in the company. The SCM team should decide which skills need to be captured. For example, knowledge of SCM procedures; experience in conducting quality audits, causal analysis, and change evaluation; and knowledge of programming languages and database management systems are all skills that could be captured in such a database.

The numbers in the columns in Table 14.1 represent the experience in years for each skill. The number of years of experience in a particular field is not always a good guide, however, to a person's knowledge level and expertise in that field. Nevertheless, it is the most easily available yardstick. Other criteria such as a merit

Table 14.1 Sample Skill Inventory Database

Name	*Thomas*	*Barbara*	*Ishtar*	*Bob*
Location/project	L1/P1	L2/P2	L2/P2	L1/P2
SCM activities	7	2	0	0
Quality audits	4	0	3	0
Causal analysis	3	4	3	0
COBOL	10	0	5	2
DB2	7	0	3	2
CICS	7	0	3	2
Oracle	0	4	0	0
Visual Basic	0	5	0	0

rating or point rating system can be used. If the company has a good personnel evaluation system, then those ratings could be used instead of the number of years of experience.

Although it is perfectly fine to store this information on an electronic spreadsheet, it would be ideal if it were stored in a relational database, because this makes the identification of the right people an easier task. For example, if you need a person to do an evaluation of a CR that involves a program developed in the COBOL-DB2-CICS environment, then you need a person who is familiar with the SCM activities and who has good knowledge of COBOL, DB2, and CICS. If you were using a spreadsheet or a manual log, it would be a very time-consuming job to locate the people you want. However, if the data were stored in a relational database, one could get an answer by simply querying the database:

```
SELECT name, Location
FROM skill_inventory_table
WHERE SCM >= 2 AND COBOL >= 2 AND DB2 >=2 AND CICS >=2;
```

This query will bring up the names of all people who have two or more years of experience in SCM, COBOL, DB2, and CICS. Then you merely choose from the resulting list.

There are some overheads in maintaining the skill inventory database. Usually, the skill inventory database is managed by the HR department. They need the list of the skills possessed by each and every employee of the company. This is required to identify the training needs of the employees, to forecast the skill requirements, and to formulate the recruitment strategy, among other things. Accordingly, the HR department usually asks employees to complete the skill inventory database and update HR on a regular basis—as and when new skills are learned. The SCM team can ask the HR department to include the list of the skills they want to be included in the list. The responsibility for managing the skill inventory database is ideal for the HR department, since this office is constantly in touch with employees and knows which employees have left and which have joined the organization. The SCM team can access the database whenever they want to get the necessary information about the people they require to perform the various SCM functions. It is important to remember that the skill inventory database is of little value if it is not kept up-to-date. As mentioned previously, people leave the company, new people join, and people learn new skills, all of which should be reflected in the database.

CCB Organization

The CCB is the apex body that decides on whether or not to carry out a change. Depending on the size, complexity, and nature of the project, there will be a single CCB, multiple CCBs, or even multilevel CCBs. In most projects, there will be only one CCB. For small projects, there may not even be a CCB, in which case, the project leader will decide whether to accept or reject a CR. For projects having a CCB, the CCB will have both permanent and ad hoc members. The permanent members include representatives of the management, project team, QA team, marketing team,

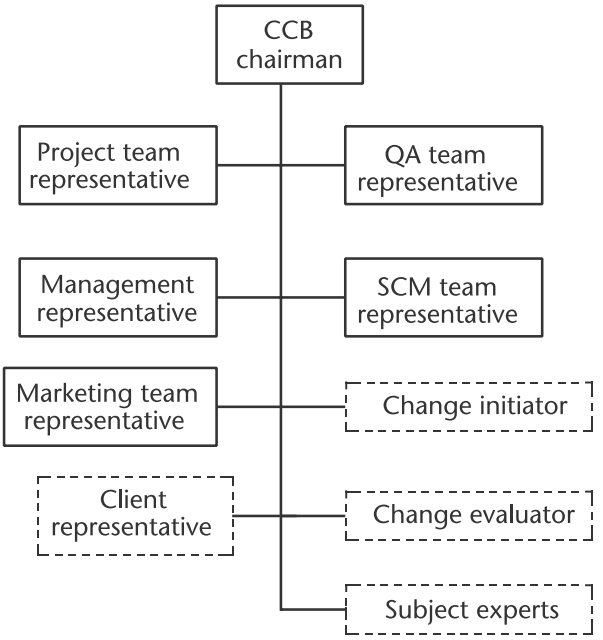


Figure 14.4 Sample CCB organization.

and SCM team and, in many cases, client representatives. The ad hoc members are people who are called (and whose expertise is required) in to resolve issues that the CCB is not able to resolve—or not able to resolve without expert advice. Sometimes the CCB may summon the change initiator or evaluator for clarifications. Figure 14.4 shows a sample structure for the CCB.

In some large and complex projects, more than one CCB will be required, with each handling different modules of the project. This situation arises when the project is very large and may be distributed geographically, so that a single CCB will not be able to handle the disposition of all the CRs and problem reports. In cases where multiple CCBs are present, there should be a "super CCB" (SCCB) or project CCB (PCCB) to oversee the functioning of the CCBs and to resolve conflicts, if any, between the CCBs. Figure 14.5 shows this type of setup.

In some cases, multilevel CCBs will be required, as shown in Figure 14.6. Here the difference is that each level of CCB will handle a particular kind of CR or problem report. For example, a level-3 CCB will handle problems with low severity. Here the CCBs are classified based on the types of CRs they handle, whereas in the case of multiple CCBs they are classified based on the module or subsystem they handle.

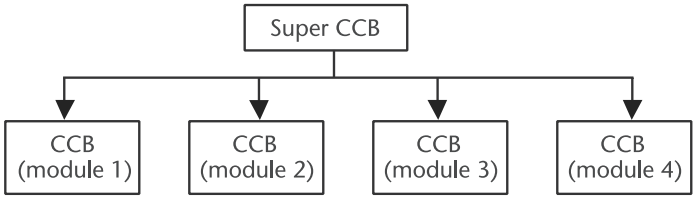


Figure 14.5 Multiple CCBs.

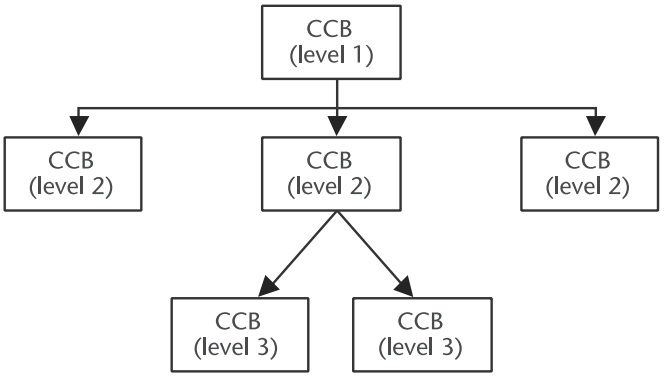


Figure 14.6 Multilevel CCBs.

So in the case of a multiple CCB environment all the CRs from a particular module will be handled by the same CCB, but in a multilevel CCB all the CRs with the same severity level will be handled by a CCB. Multiple CCBs are useful when the project is large and has many modules, and development takes place in different locations. Multilevel CCBs are applicable to large projects that are complex in nature, but the development occurs in one place. Here the multiple levels help to resolve simple problems faster because the lower-level CCB would be comprised of the module leader and an SCM representative, who can make decisions faster rather than waiting for the full CCB meeting to happen. Thus, the load on higher-level CCBs is reduced, and they can focus their attention on critical and severe problems that will have huge impacts on costs and project schedules.

Summary

The place of the SCM team in the organizational structure depends very much on the organization and varies from one organization to another. Irrespective of the position in the organizational hierarchy, however, the SCM team performs the same functions.

We have examined the organization of the SCM team and the different people who play significant roles in the functioning of the SCM activities. We have also considered the different types of CCBs and how they function.

Because SCM tools are automating more and more tasks, the role of the SCM team and the tasks it has to perform are being reduced. As a result, in the future, we may see reductions in the size of the SCM team, except in some areas such as analysis, evaluation, and audits, where human intervention is still required.

Reference

[1] *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610–1990), IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition)*, Piscataway, NJ: IEEE, 2003.

Selected Bibliography

Alain Abran, A., and Moore, J. W., (Eds.), *SWEBOK: Guide to the Software Engineering Body of Knowledge (Trial Version)*, Los Alamitos, California: IEEE Computer Society, 2001.

Ben-Menachem, M., *Software Configuration Management Guidebook*, New York: McGraw-Hill, 1994.

Bourque, P., and R. E. Fairley (eds.), *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition), Piscataway, NJ: IEEE, 2003.

Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

CHAPTER 15

SCM Tools

Introduction

SCM tools are becoming more important in today's complex software development environments. Today a typical software development project consists of multidisciplinary teams, spread across different parts of the globe and working in networked environments and different time zones. To avoid chaos and to bring in discipline and improve development productivity, the role played by SCM tools is becoming more and more important.

SCM tools are nothing new. They have existed in the mainframe and UNIX environments for many years, and now they are available for every platform and every kind of development environment. In fact, hundreds of SCM tools are available in the marketplace.

It is important to remember, however, that no SCM tool is a panacea for all SCM problems. The purchase of a sophisticated tool is just one step toward effective SCM. Using the wrong tool, or using the right tool ineptly or insensitively, makes SCM problems worse, not better [1]. This chapter examines SCM tools and considers what they can do to automate the SCM process and improve development productivity. We also discuss how to select an SCM tool that is best for your project or organization and take a look at the question of whether to make or buy your organization's SCM tools.

Evolution of SCM Tools

Initially, SCM was just a means of controlling changes to the source code. Tools like SCCS and revision control system (RCS) under UNIX were created for that purpose, but their function was quite limited. Eventually, the ability to manage revisions, compress deltas, handle binary file types, and increase security levels was added—and source code control turned into version control.

SCM evolved into a more comprehensive process as developers realized the need to track more than just changes to the source code. Today, SCM tools include such diverse features as build management, defect and enhancement tracking, requirement tracking, release management, software production control, software packaging and distribution control (the licensing and generation of serial numbers, CD keys, and codes), and site management. The ability to identify components, recreate deliverables, monitor problems and change requirements, and deliver a product of

consistently high quality is the current goal of SCM. Thus, it involves monitoring every phase of a software product's life cycle.

SCM tools have come a long way from just managing the source code. They are now responsible for one of the critical functions of software engineering and help to keep projects on track and manage them effectively. Today's SCM tools are getting bigger, better, and bolder. They are bigger because they no longer manage just code; they can manage the development of any object in the system's life cycle. They are better because they support parallel as well as incremental development. They are bolder because they have expanded their functionality beyond the control and management of objects and into the control and management of processes [2].

Reasons for the Increasing Popularity of SCM Tools

There is no question that the market for SCM tools is very hot. Industry analysts are forecasting steady growth rates for the SCM tools market. Why have so many companies replaced their manual or semiautomated SCM systems with SCM tools? Why are so many companies, which were using old SCM tools like SCCS or RCS, changing to more sophisticated tools? Here are some reasons:

- *Development time reduction.* SCM tools reduce development time by improving development productivity and reducing mistakes.
- *Increased business agility.* Improved productivity and reduced development time means less time is needed to get a product to market. This coupled with the SCM tool's ability to manage and track problems; fix them; rebuild the systems easily, accurately, and quickly; and release them faster results in improved customer satisfaction. So a company can be more agile and more responsive to the needs of the customer without compromising product quality.
- *Error reduction.* SCM tools automate most of the monotonous and repetitive tasks that were previously done by people. Thus SCM tools have greatly reduced the opportunity for human error.
- *Information integration.* One of the major functions of SCM is to provide sufficient, relevant, and accurate information about the software system to the different people in the project—such as programmers, managers, analysts, and auditors—allowing the software development and maintenance processes to proceed smoothly. Earlier, people had to rely on reports provided by the status accounting function. These reports were not capable of answering ad hoc queries. However, today's SCM tools have all the information users need and can deliver them to users in any format that they want almost instantly. This information integration capability and flexibility is one of the most important advantages of SCM tools.
- *Automation.* SCM tools automate many processes, including configuration control, defect tracking, status reporting, version control, and build management. These were the tasks that were considered the "necessary evils" of manual SCM systems. By automating these tasks, SCM tools have improved development productivity and given people more time for the system development process.

These are some of the reasons for the increasing popularity of the SCM tools market. As more and more companies join the race, the competition is getting hotter, and SCM tool vendors are gearing up to meet the competition by offering more features and better capabilities for their products. So the future will see a fierce battle for market share and mergers and acquisitions aimed at gaining strategic and competitive advantage. The ultimate winner in this race will be customers, who will get better products and better services at affordable prices.

Advantages of SCM Tools

Installing an SCM tool has many advantages, both indirect and direct. The indirect advantages include a better corporate image, improved customer goodwill, and customer satisfaction. The direct advantages include use of the latest technology, flexibility, better analysis and planning capabilities, improved development efficiency (people will have more time to work on developmental activities), information integration for better decision making, and faster response time to customer queries. Some of these direct benefits are described in the following sections.

Information Integration

The first and most important advantage lies in the promotion of integration. The reason SCM tools are called integrated is because they have the ability to update data automatically between related SCM components. For example, it is only necessary to update the status of a CR at one place, for example, in the change management system; all the other components will automatically get updated. So when a developer checks in a component after making changes, the status of a CR automatically changes from "assigned" to "complete." The new files or files that are checked in are automatically assigned the new version numbers. The dependency details are updated to reflect the new change. The new versions are designated as the latest versions of the items and so on. The next time a status report is created all of these modifications will be reflected in that report.

The beauty of this system is that information updating happens instantaneously. So you get up-to-the-minute information at your fingertips. This information integration leads to better decision-making and resolution of problems.

Another advantage of this integration is that the people who are involved in a project are also connected to each other. This integration has tremendous potential for improving productivity. As a result, one can have, for example, virtual CCB meetings, on-line polls, and automatic notification.

Flexibility

A further advantage of SCM tools is their flexibility. Diverse multinational environments are covered in one system, and functions that comprehensively manage multiple locations and distributed and parallel development can be easily implemented. To cope with globalization, distributed development, and sophisticated development projects (development of large, complex, and mission-critical systems), this flexibility

is essential. Moreover, it has major advantages, not simply for development and maintenance, but also for management.

Better Analysis and Planning Capabilities

Another advantage provided by SCM tools is the boost to planning functions. By enabling the comprehensive and unified management of related SCM functions (such as configuration control, status accounting, and CAs) and their data, it becomes possible to utilize fully many types planning tools, such as decision support systems and simulation functions and what-if analyses. For example, we could simulate the impact on the project cost and schedule of using more than one person on a change implementation. Furthermore, it becomes possible to carry out flexibly and in real time the filing and analysis of data from a variety of dimensions. Decision makers can get the information they want and, as a result, make better and more informed decisions.

Use of the Latest Technology

An additional advantage is the utilization of the latest developments in IT. SCM tool vendors realized that in order to grow and to sustain that growth, they would have to embrace the latest developments in the field of IT. Accordingly, they quickly adapted their systems to take advantage of the latest technologies such as open systems, client-server technology, and the Internet and intranets. It is this quick adaptation to the latest changes in IT that makes flexible adaptation to changes in future development environments possible. It is this flexibility that makes the incorporation of the latest technology possible during the system design, development, and maintenance phases.

Why Many SCM Tool Implementations Fail

This section examines why many SCM tool implementations fail. SCM tools—if chosen correctly, implemented judiciously and used efficiently—will raise productivity, shorten development times, improve responsiveness, and result in better customer satisfaction. To use a tool efficiently, it has to be the right tool (right in the sense that it integrates well with the organization's business processes), and the people who use the tool have to be properly trained. However, many projects or organizations fail, because they use the wrong product, conduct an incompetent and haphazard implementation, and suffer inefficient or ineffective usage.

SCM is first an attitude; second, a process; and only third, a set of tools [3]. Attitude refers to the feeling or mood of the people in the organization toward SCM. Users of the SCM system have to be convinced of the real benefits of using SCM. This can be done by educating staff about SCM's benefits and exposing the misconceptions about SCM. This is important because, if people believe SCM is unnecessary, they will try to bypass it—and unless a consistent SCM process is

integrated into the development methodology, no one will know when to apply SCM tools, or even what those tools are.

To work successfully, SCM solutions need a lot of factors to click. There should be good people who know the SCM concepts and the project and organizational details, the vendor should be good, and the vendor's package should be the one best suited for the company's needs. The implementation should be planned well and executed perfectly, and the end-user training should be done so that the end-users understand the system and the effect of their efforts on the overall success of the program.

SCM can be implemented in a variety of ways. It can be implemented in a phased manner (the incremental approach) or it can be implemented in one shot (the big-bang approach). Irrespective of the implementation methodology chosen, the development process has to be mature, and the development environment and the organizational culture must be conducive for the implementation of SCM and its tools. If the implementation is done in an environment where personnel are not ready for SCM and the work culture is not suited, then the implementation will fail. Accordingly, the first step in the SCM implementation process is to make the organizational environment suitable for SCM.

Organizations exist in different levels of process maturity. The ease and correctness with which they can execute a process depends on their maturity. As such, it is generally not fruitful to impose a very sophisticated process on an organization whose maturity is low. The maturity of an organization not only depends on the skill sets of the individuals, but also on the chemistry of the team. This makes improvement an incremental process. The lessons, habits, and practices learned doing simple tasks provide the foundation to take the organization toward more sophisticated tasks [4]. So the maturity level of the organization is very critical in the success of an SCM implementation.

Also, many SCM tool implementations fail because they try to implement SCM in the "tool" way rather than in the "process" way. As discussed earlier, SCM is first an attitude, then a process, and only third a set of tools. One can implement sophisticated, all-encompassing SCM tools, loaded with features, but if the people who are using them do not know why they are doing a particular task or what happens when they do something [or the effect of their actions (or lack of it) on others], then the chances of such an implementation succeeding are slim.

Thus, for an SCM tool implementation to succeed, the implementation has to be planned well, people must be ready for SCM, the development process must be in place, the selected tool should be well integrated with the process, and finally everybody involved in the project should understand the importance of SCM and the role each has to play to make the implementation a success.

SCM Tools and SCM Functions

The days when a single programmer developed an application independently are a thing of the past. Today's software development is a team effort. When more than one person is involved in the development of a software system, it can lead to such

complications as communications breakdowns, shared data problems, and simultaneous update problems. We have discussed how SCM systems can help solve these problems. Manual SCM is a tedious, monotonous, and time-consuming process. Many tasks like change management, record keeping, and status reporting are repetitive in nature. Accordingly, we need to automate these tasks: A good SCM tool should automate the project coordination and management tasks, support repeatable processes, manage changes and issues, and automate the system builds.

In a manual SCM scenario, a developer or programmer spends more time on nondevelopmental activities such as documentation, modification, and revision of the requirements and design documents, and bug fixing. Another portion of the nonprogramming time is spent on such tasks as handling CRs, tracking user requirements, doing causal analysis, and creating status reports. Even though all of these mundane tasks are necessary for the smooth functioning of a project, it is a cardinal offense to waste the precious time of programmers and developers on them; such jobs can easily be automated. This is where automated SCM tools can be invaluable.

SCM tools automate manual tasks and set development teams free to actually develop. Moreover, today's software systems are more sophisticated and highly complex, support multiple platforms (sometimes they are cross-platform), use complex technologies, and are developed in distributed environments by diverse development teams (with different cultural, social, educational, and ethnic backgrounds). In fact, managing projects of this kind manually is almost impossible. Consequently, SCM tools are no longer a luxury, but a necessity. Not having an SCM tool can become a strategic disadvantage and can lead to catastrophic consequences such as failed projects, cost and time overruns, and customer dissatisfaction.

The following sections examine the features of SCM tools that help automate the SCM functions, reducing human intervention and improving productivity.

Version Management

Version management, a critical function of SCM, is the basis on which other functions are built. Version management is the storage of multiple images of the development files or set of related files in an application. With a good version management system one can get an image—a snapshot in time of the development process—and recreate the file or files as they were at that discrete point in time.

Any good SCM tool will support version management activities such as creating, working, and changing versions. A version consists of a file or a set of files, each with a particular version label (i.e., a unique identification or name). The SCM tools have the capability to define what file or files make up a version. Once a version is defined, the user is able to check out the files that make up a version. The SCM tool is able to identify the changes to the components of a version and then create or define a new version. In this area—identifying the changes and creating new versions—and in the system building (or build management) area, the version management system works in conjunction with the change management system and build management system so that the correct versions are incorporated into the system builds.

Change Management

Most SCM tools completely automate the change control procedures and manage the repositories. When a CR is made, the information about it goes to all the concerned personnel (e.g., management, project leads, and CCB members) so that they can send their approval or disapproval immediately by e-mail or some other messaging system. This eliminates the time lag between change initiation and disposition. Virtual CCB meetings and on-line polls are now standard with almost any good SCM tool. The change management system can be configured (a rule-based system) to automatically receive the CRs, process them based on a set of rules, get the responses from the appropriate sources regarding the disposition, and allocate the implementation work to qualified professionals and notify them regarding the work allocated to them. All of these things can be done without human intervention. This kind of workflow automation helps to improve productivity and shorten the change process. Also, when a developer checks out the file(s)[1], the change management systems will begin the tracking of the activities that affect those file(s).

The change management system is also notified when the task of implementing a change is allotted to someone. So only authorized personnel can check out or make changes to the specific files. The change management system keeps track of the modifications made to the change set and its components, and when the change is completed and the items are checked in, the system can compare the before and after images and create the change history and the delta and store the items in the most efficient manner. Also, the details of the change process, such as who initiated the change, which people were involved in the decision making, who implemented the changes, how much time was taken for implementing the change, and how the change was implemented are captured automatically. This information is relevant to the status accounting function and helps in managing the project more effectively.

We have seen that the configuration identification function is an element of CM, consisting of selecting the CIs for a system and recording their functional and physical characteristics in technical documentation. In an automated environment, this type of information can be captured and updated automatically. The parsing tools can determine the interdependencies of the various CIs so that when shared components get modified, the system can alert users about other impacted items. This automatic capturing of the component interdependencies saves a lot of time that would otherwise be spent on impact analysis when a CR is initiated. With this information already in the system, users just have to query the system repository to determine the impacted items.

Most change management systems support team development, parallel development, and distributed development. Many of the modern tools allow more than one person to work on the same file or set of files. In parallel development, two or more users may need to work on the same file, just as they would do in concurrent

1. A person can check out more than one file, because a CR implementation is rarely confined to a single file; even if it is confined to a single file, information such as the associated analysis and design documentation and user manuals also have to be updated.

development. In the case of concurrent development, the branches are eventually merged into a single item, whereas in the case of parallel development, the branches will go ahead without merging. Parallel development is necessary when the development teams are producing versions of the same component for different hardware platforms or operating systems (also called variants).

Multiple development paths are usually supported with branches. Branching allows users to store more than one path for the same file. In the case of concurrent development, once the changes are made and the different people check in their files, the system will compare the changes and do an automatic merging of those changes. There is also the facility to do interactive merging, where the system will highlight the changes made by the different people with respect to the ancestor object. The systems can be configured to do automatic merging or interactive merging depending on the user's preferences. There are certain pitfalls associated with automatic merging, which can fail at times. The cases where automatic merging might work are where two users edit the same file and make changes at different locations in the file. Then, perhaps, the SCM tools can merge the changes. However, even this simple looking case can produce bugs. The overall resultant file might have a logic flow that is flawed. Similarly, there is the case where two users edit the same section of the file where automatic merging cannot be done.

Change management systems keep a chronological record (or journal) of all activities that are applied to the system components so that at any point any object or component in the system can be brought back to any of the previous states. This is helpful in cases where you need to unwind the impact of an emergency fix that was done in the middle of the night, so that it can be thoroughly checked, tested, and repromoted.

Another important and very useful feature of SCM tools is their graphical interface. The tools use different colors, icons, and other features to help users absorb information quickly and determine patterns and exceptions easily.

Problem Tracking

Problem reporting and tracking is one of the activities that takes a lot of time and effort. Problems have to be reported, analyzed, and fixed, and defect prevention methods have to be carried out. The existing problem records (the knowledge base) can be scanned for similar incidents that may have been solved already. This searching is best done by a computer, especially when we are talking about large knowledge bases. Computers can easily find the records that match the search criteria—in a fraction of the time taken by a manual process.

The problem tracking components of SCM tools track the problem from its origin to completion and capture the details of the problem, such as originator, date of problem identification, cause of the problem, how and when it was fixed, how much time was taken, and what kind of skills were needed. These details are captured automatically as the activities happen. Modern problem tracking tools are very sophisticated and have advanced features like automatic receiving of the PRs, automatic categorization of the reports, rule-based actions, automatic notifying, and alerting mechanisms.

Many tools are capable of automatically notifying the concerned personnel by e-mail or pager messages about the arrival of a PR depending on predefined criteria chosen, such as severity and impact. These tools are also capable of creating alerts (automatic notification and problem escalation) based on the promotion of the problem through stages of resolution.

Promotion Management

As software systems are developed, they go through the SDLC process. Depending on the life cycle model chosen, the phases will differ, but all software components have to go through various phases such as analysis, design, development, test, release, and maintenance. Promotion management tools automatically record the phases that the CIs in the system go through and the various details such as when each phase started and when it was finished. They capture information and create trails, which will be very useful when one needs to know exactly what happened or needs to recreate an event or an item before or after a particular event. For example, if we know exactly what was done just after alpha testing, we can recreate an item so that it is in the state it was in before the test, when it did not have any problems.

Obtaining this information and these facilities are possible in manual systems also, but with an SCM tool the degree of detail that can be captured is almost infinite.

System Building

When the components of a software system are ready to be tested or shipped, we do what is called system building. We combine all files and may compile them, link edit them, and create an application. This build can be done for a subsystem, a module, or an entire system, in support of integration testing, alpha or beta testing, and system release. At this stage, capturing the details of each and every build and the building process—such as which components were used in the build, what versions of the components were used, which operating system was used, which version of the operating system, and what compiler and linker options were in effect—is of paramount importance. This is because each build should be reproducible and repeatable, and that reproduction has to be reliable and accurate. SCM must be able to recreate, for example, the alpha test version of the system perhaps one year after the actual testing was done. To do this, one needs the information mentioned above.

The SCM build management tools capture this information (in most cases automatically) and help provide reliable, repeatable builds. The automatic source code scanning, the dependency analysis capabilities, and the creation of build audit trails (footprinting) are just some of the features of the SCM tools that save time, shorten build cycles, and eliminate build errors by providing repeatable, automated builds of software projects. The SCM tools make the cross-platform builds that span multiple platforms, operating systems, and development environments easier, faster, and more accurate.

Also, many of these systems maintain the history of the previous builds and releases in their repositories. This helps when monitoring conflicts between the new releases and any of the previous releases that are still in use.

Status Accounting (Querying and Reporting)

Status accounting consists of the recording and reporting of information needed to manage a configuration efficiently. With manual systems, one has to record all events that will be needed later. This record keeping is a tedious and monotonous job and requires effort by the many people who perform the different SCM functions. There is also a limit to which the details can be captured. Also, the manual reporting function basically relies on routine reports to satisfy the information needs of the various participants in the software development process. Ad hoc queries are always time-consuming and difficult to accommodate.

When one uses an SCM tool, however, as we have seen earlier, information integration occurs. There is no need for active record keeping; the system monitors all activities and keeps a record of them to the level of detail specified by the user. Once the information is in the SCM tool's database, it can be managed at will. The powerful querying tools allow users to get accurate answers to their numerous and varied information needs, when they need them. The reports can be generated in any format specified by the user. Graphical interfaces and templates or wizards are available to create a query and get the answers. This accurate and immediate access to information is one of the most often developer-requested features of SCM automation.

CAs

Auditing means validating the completeness of a product and maintaining consistency among the components by ensuring that the product is a well-defined collection of components. For auditing requirements, one needs a history of all changes, traceability between all related components in the product and their evolution, and a log containing all details about work done [5]. The reporting features of the SCM tools provide this information.

The reporting feature of the SCM tools identifies the differences between the versions and releases. The ad hoc query facility of the tool helps answer any specific questions that the auditor has. The SCM tools automatically record all activities happening to the CIs and thus provide a comprehensive audit trail of the activities performed on an item and the events that have happened. These reports and logs help a great deal in automating the auditing process and make the CAs a lot easier than the manual process.

Access and Security

The information contained in the SCM tool is sensitive and should not be available to all. Also, the items in the repository should be accessible only to those who have the necessary authorization. In the case of manual systems, configuration managers will have to take action to prevent unauthorized access to files or information. However, the SCM tools have features that aid in managing access to the system.

The system can be configured so that only people with the necessary authorization will get to see the information or have access to the files. These access privileges

and security mechanisms can be enforced using the user ID and login password of the users, so that the exact mechanism is transparent to them.

Users can log in to the system and access all information they are authorized to see, and what they are authorized to see can be set based on the user ID (or designation, title, role). With distributed development scenarios, where developers access the SCM information and files using the Internet, encryption methods are used to ensure secure transmission of data and information across the Internet.

Customization

The era when you had to buy a tool and implement it as it came out of the box is over. SCM tool vendors are now offering the ability to customize their tools. The extent of the customizations varies from tool to tool. SCM tools are being enhanced with customization facilities so that customers can easily modify certain features of the tools (such as screen layout, colors, and state names, and transitions). More complex customizations (such as changing the associated semantics, roles, access rights, and transition conditions of the states) typically require source code changes or additional scripts using triggers and event mechanisms. Vendors are moving toward parameterizing these more complex customizations [6].

These customization capabilities will help customers, because they will be able to get tools that perfectly match their requirements. Thus, instead of developing a process around the tool, now customers can choose a tool that will integrate seamlessly with their development process.

Web Enabling

As with every other software market, SCM tool vendors are being forced to move from a client-server to browser-server architecture to Web-enable their tools. The popularity of distributed development makes Web enabling a must-have feature in SCM tools. Today's developers need to access the corporate databases and repositories when they are on the move and when they are working away from the office. Also, different people working on the same project need to share central databases and repositories for performing the various SCM activities. The most cost-effective way to do this is to use the Web. The availability of high-speed Internet access and technologies such as virtual private networks (VPNs) and better encryption methods make the use of the Internet a very cost-effective and secure medium in which to do distributed development.

The previous section discusses the general characteristics of modern SCM tools and their advantages. There are more than 50 SCM tool vendors in the marketplace, and new players are entering the market. A list of the major SCM tools is provided in Appendix A.

As described earlier, SCM tool selection is one of the critical factors for the success of SCM implementation. The success of a company lies in selecting an SCM tool that suits its needs and matches its profile. The next section examines how to select the right SCM tool for your organization or project.

SCM Tool Selection

SCM tools have gained popularity, and their usefulness has increased to a point where software development without an SCM tool is almost nonexistent. Furthermore, manual SCM systems are not acceptable, except for very small, single-person projects. All other projects benefit from the use of automated SCM tools.

SCM tools are now available in all sizes and shapes for all platforms and development environments. Evaluating the SCM tools available in the marketplace and then selecting one for your organization or project are critical parts of the process. This decision can make or break an organization.

Of the more than 50 SCM tools available, the features they offer vary, as do the technologies they support, the technologies they use, the architecture on which they are built, and their available platforms. Each tool has its own strengths and weaknesses. For example, some are better at change management, whereas others have excellent build management and versioning capabilities. There are SCM tools that cover the entire spectrum of SCM functionality, and there are tools that just do source control.

Deciding which tool is suited to your organization is a difficult task. Each piece of marketing literature of the tool vendors claims that their product is the best among the lot and has all of the features you will ever need. So, if you go by what is written in the product brochure or what the salespeople say, you will find it very difficult to make a decision and might end up with the wrong choice. According to Dart [6], "… such literature (the marketing literature of the tool vendor) is valuable for giving the reader an overview of functionality and a glimpse at the differentiator for that vendor's offering. But, if you compare the literature or listen to a vendor's presentation, it would be very difficult to evaluate which package is the best or which would be most suitable for your organization." Accordingly, tool selection is something that should be done in a systematic and scientific manner. This section examines how to select an SCM tool that will suit your needs.

The most important factor to keep in mind when analyzing the different packages is that none of them is perfect, and this needs to be understood by everyone on the decision making team. The objective of the selection process is not to identify a package that covers each and every requirement (a perfect fit). The objective is to find a package that is flexible enough to meet the company's needs—or in other words, to find a tool that can be customized to obtain a "good fit." Because there are so many, analyzing all SCM packages before reaching a decision is not a viable solution. It is also a very time-consuming process. So it is better to limit the number of packages that is evaluated to less than five. It is always better to do a thorough and detailed evaluation of a smaller number of packages than to do a superficial analysis of dozens of packages. So the company should do a preevaluation screening to limit the number of packages that is to be evaluated by the committee. The preevaluation process should eliminate those packages that are not at all suitable for the company's business processes. One can zero in on the few best packages by looking at the product literature of the vendors, getting help from external consultants, and, most importantly, finding out what package is used by similar companies. It is a good idea to look around to find out how the different packages are performing in environments similar to yours. Once you select a few

packages after the screening, you can call the respective vendors to request presentations or demos.

Dart [6] classifies SCM tools into three categories: version control tools, developer-oriented tools, and process-oriented tools. The SCM tool evaluation process can be narrowed down to a few tools if the company knows which category of tools it is looking for. The tools vary in features, complexity, and functionality, with the process-oriented tools being the most sophisticated and having the most functionality.

A version control tool would typically suit a small company or a research and development group that has a small number of releases and possibly no variant releases. A developer-oriented tool would typically suit a medium- or large-sized company that does not have a lot of formal processes defined and is not focused on standards certification. The company might have many variant releases and need strong support for parallel development and build management, as well as more reliability from the CM repository. A process-oriented tool would typically suit a large corporation with formal processes that need to be automated, that is focused on process improvement in general, and that has sophisticated build management and change management needs [6]. The preceding categorization of SCM tools can be used to narrow the list of candidates in the preevaluation screening process.

After the decision to buy an SCM tool has been made, the company needs to develop selection criteria that will permit the evaluation of all the chosen packages on the same scale. To choose the best system, the company should identify the system that meets the business needs, that matches the development process, and that identifies with the development practices of the company. It will be impossible to get a system that performs its development and maintenance functions in exactly the same way as the company does, but the goal is to get the system that has the least number of differences.

Selection Process

The selection process is one of the most important phases of SCM implementation, because the tool that you select will decide the success or failure of the project. Because SCM tools involve a huge investment, once a package is purchased, it is not an easy task to switch to another one. So it is a "do it right the first time" proposition. There is little to no room for error.

Selection Committee

It is a good idea to form a selection or evaluation committee to do the evaluation process. The selection committee should be entrusted with the task of choosing a package for the company. The package experts or consultants can act as mediators or play the role of explaining the pros and cons of each package.

The evaluation committee should be made up of various representatives of the user community [7]. It can include developers, testers, QA people, technical leaders, build managers, and project managers. All provide perspective and ensure their needs are addressed, while providing their own experiences, skill set, and processes to address the three important areas apart from functionality requirements: usability, performance, and scalability requirements.

Working with Vendors

Once you decide to buy an SCM tool, the marketing executives of the different vendors will swamp you. Each will have colorful and superbly produced brochures and presentations claiming that their product is the best one for you. They will try eagerly to convince you of that. Accordingly, you should have a strategy in place for working with these vendors.

As mentioned, you should conduct a detailed evaluation of not more than five packages that meet your preselection criteria. When the vendors arrive for their presentations, you should be thoroughly prepared; otherwise they may overwhelm you with their presentations and you will not have time ask questions. This point is being stressed again and again because most vendors are able to make presentations that leave potential users dazzled, and without proper consideration of all aspects, the selection may end up being based on a set of factors that is insufficient for arriving at a well-informed and judicious decision.

So instead of just listening to presentations, you should be prepared to ask questions, prepared beforehand and addressing all of your concerns. The responses that you get to your questions will help you either eliminate a vendor or strengthen his or her case. The questions, if properly prepared and asked, will expose the weak or problem areas, if any, that exist in the vendors' products. Also, when you are asking questions, it means that you are not taking anything for granted. It is a good idea to prepare minutes of the meeting and ask the vendors to sign off on them. Another way to do this is to send a detailed e-mail after the meeting, to the vendor, with the points that were discussed. Ask for e-mail feedback to put on the record. These procedures will prevent vendors from making false claims, and you can make them accountable if they fail to deliver what they have promised.

The vendors should be asked to show testimonials and practical demonstrations of the system. In addition, they should provide references for organizations where the system has been implemented successfully. However, all vendors will also have customers where the tools have failed. Getting those names and reasons for the failure is more important than the success stories. While vendor representatives are well prepared to tell their success stories, questions about failed implementations usually reveal points and issues that they are trying to downplay. So it is important to ask about failed implementations. Quite often, the vendor will send two representatives to visit you, a marketing agent and a technical expert. Most of your questions should be directed to the technical expert. The marketing expert should be asked about such factors as warranties, licenses, cost, support, and training, whereas the technical expert should be asked about the functionality and capabilities of the system they are offering.

Role of Technology

Existing technology will play a very important role in the SCM tool selection process. Each organization has its own technological environment (e.g., how the development process works, what kind of hardware and software it uses, and a preferred

database management system). These factors can greatly influence the selection process in the sense that they can limit the number of packages available for evaluation. So management must decide whether the SCM tool will be selected taking the existing infrastructure into consideration or whether the existing systems will not be considered (in which case some of them will have to be scrapped). This is a hard decision, and it is always a better idea to find a package that is compatible with the hardware, software, and technology the company already has in place. Also, if the organization has the necessary infrastructure, then it can think of buying the required components from the vendors and integrating them with the existing system.

For example, if an organization is using the operating system's library management system and is quite satisfied with it, then it can go in for a change management and problem-tracking tool and not the complete offering from the vendor. Later, if the organization wants to switch from the operating system's library management, it can purchase the remaining modules of the package. So it is not imperative to buy all of the components offered by the vendor. The evaluation committee in association with the vendor can select the required components and then integrate them with the existing infrastructure. However, do not forget here to get the vendor's assurance (in writing) that the existing system will integrate smoothly and seamlessly with the purchased components.

Selection Criteria

SCM tools come in all sizes and shapes, with all of the frills, bells and whistles, gizmos, and gadgets that you can imagine. So it is a good practice to specify selection criteria for evaluating the packages that survive the preevaluation screening. The criteria can be in the form of a questionnaire, and a point system can be implemented. This will help make the selection process more objective.

The questions should address the organization's needs and concerns, and each issue or question should be given a weight according to how critical that function is for the company. For example, if the company is doing distributed development and has development centers in different countries, then the ability to handle distributed development and Web features becomes an important criterion. Likewise, the selection criteria should be divided into categories—vital, essential, and desirable—and points should be given to each criterion. This point rating system simplifies the evaluation process, but remember that the importance of human intuition (gut feeling) and judgment should never be underestimated.

The best method for preparing the selection criteria is to conduct a requirements analysis—find out what the company needs. The requirements must reflect those factors that the company considers indispensable for the successful running of the business according to the company's work culture and practices. A set of questions that could form part of the selection criteria can be found in the following documents:

- Mosley, V., et al., "Software Configuration Management Tools: Getting Bigger, Better, and Bolder," *Crosstalk: The Journal of Defense Software Engineering*, Vol. 9, No. 1, Jan. 1996, pp. 6–10.

- Firth, R., et al., "A Guide to the Classification and Assessment of Software Engineering Tools," Technical Report CMU/SEI-87-TR-10, Software Engineering Institute, Carnegie Mellon University, 1987.
- Berlack, R. H., *Evaluation and Selection of Automated Configuration Management Tools*, Amherst, NH: Configuration Management International, 1995.

Some examples of selection criteria are listed as follows:

- The package should have distributed development support.
- The package should support parallel development of variants.
- The package should have both automatic and interactive merging facilities.
- The package should have a a customizable report generation facility and the facility to export the reports to other systems.
- The tool should support a footprinting feature for build management.
- The change management and problem tracking system should have the facility to carry out communication such as virtual CCB meetings, on-line polls, and automatic notification.
- The system should have a graphical user interface.
- The performance of the system should be within in such-and-such limits.
- The vendor should have been in the business for at least "x" years.
- The package should have at least "x" number of installations out of which at least "y" should be in organizations similar to your organization.
- The cost of the package with all the necessary modules should be less than "x" dollars.
- The package should support incremental module addition. For example, the company should be able to buy the core modules initially and then purchase additional modules as and when desired.
- The vendor should provide implementation and postimplementation support.
- The vendor should train company employees on the package.
- The package must be customizable and the customization process should be easy (something that can be done in-house).
- The package should be scalable or should be capable of growing with the organization.
- The vendor's policy and practices regarding changes such as updates and versions should be acceptable.

In this way, the issues, concerns, and expectations that the company has regarding the package can be consolidated and made into a list. Then the items in the list should be divided into the "vital-essential-desirable" categories. Subsequently, using this list, each package should be evaluated. Many items in the list will have descriptive answers. The committee should analyze these issues and assign points to them.

One important thing to keep in mind is that whenever a decision is made, the committee should discuss it and reach a consensus. In doing so, the chances of conflict between different functions (like the development team, QA team, and other support teams) are reduced. Remember that the SCM tool belongs to all functions,

so it is better for decisions to be arrived at via a consensus. This will create the notion that the tool belongs to everyone, and it furthers the idea that a commitment from everyone is needed to make it happen. Most importantly, because both the SCM experts (people who know the tools well) and project team members (people who know the project and work culture well) are involved, they can point out areas and issues that should be given more importance and the aspects that should be scrutinized more thoroughly.

Another source from which the evaluation committee can get information about the tools is independent research agencies and companies. These sources supply information, comprehensive analyses, and comparison reports about the leading tools. It is important to remember, however, that these reports, although excellent sources of information and a single-point reference about the leading SCM tools, are not totally unbiased, completely accurate, and totally objective. Therefore, they should not be taken as gospel truth. Still, these reports can provide valuable information about the tools. Accordingly, to get a complete picture of the SCM tool marketplace, study at least a few reports by these research groups along with the vendor's literature. These reports analyze and compare the tools and their features, predict market trends, and forecast the position of the different players in the coming years, among other things. A number of companies and consultants do this kind of analysis. Prominent among them are Ovum Limited (http://www.ovum.com), the International Data Corporation (http://www.idc.com), and the Butler Group (http://www.butlergroup.com). Sometimes trade magazines like *Communications of ACM, IEEE Software, IEEE Computer*, and *Application Development Trends* publish articles about SCM and its current state. This information is also worth looking into because it is independent and not biased. In addition, Web forums can give information about SCM tools and their pros and cons. These forums are run by SCM professionals, and most SCM practitioners are members of them. Their members range in experience from newbies to veterans who have worked on many SCM tools and have spent years in the field. It is possible to get a lot of information from these forums, and if you don't get what you are looking for, you can always ask and your query will be answered by experts. Tapping the potential of the worldwide SCM community through the forums for tool selection (and for finding answers to other questions) is a very cost-effective (as the membership to these forums is free) and efficient way to select the SCM tool best suited for your organization and then operating and maintaining it.

Once the committee has evaluated all the tools that have cleared the preevaluation stage, listened to the vendor presentations and demos, and cleared pending issues, it reaches a decision on which tool to buy. At that point, it is a good idea to visit a few companies that have installed the particular package to see it in action. Since many people will not admit when they have made a mistake, anything the existing owners say about a package should be taken with a grain of salt. Nevertheless, visiting four or five installations should give the committee members a good idea about the package. These company visits may not be practical for small firms and, therefore, should be done only if you have the budget for such an activity and if the team members feel the need. If the committee members feel that their decision is right, then the company can proceed with the purchase and implementation of the chosen tool.

If anybody is uneasy about some aspect or does not feel that the product meets the expected standards, then the committee members should revisit the question of which tool to choose and be prepared to do the analysis once again. The package that received the maximum score in the point rating system need not be the one that is best suited for the company. Accordingly, the extra time spent on analysis and evaluation is not a waste; in fact, it could save the company from a potential disaster.

Tool Implementation

Once the right tool has been selected, the next step is implementation. Making a major change in a company, such as changing over to a new automated CM tool, is both a significant opportunity and a major responsibility [8]. It is an opportunity because it enables a company to address its CM problems and improve processes to result in better management of its data and its development and maintenance activities. At the same time, changing over is a major responsibility because of the ramifications and the resources required to make the change. Many tricky technical, political, organizational, cultural, process-oriented, risk-related, and personnel issues need to be addressed in making the change, and people need to be committed to the change.

Accordingly, to adopt an SCM tool successfully or to make the changeover from one tool to another, careful planning is a must. All possible aspects of the implementation should be addressed satisfactorily before starting the implementation or changeover. This includes the implementation of new procedures; the changing responsibilities of users (with SCM tools, users get more freedom than with a manual system, like the ability to check out and check in files and interactively merge the changes); the changing responsibilities of the SCM team (because most of the processes become automated, a significant reduction is seen in the size of the team); and the role of the CCB members, QA members, and the audit team (they will have to be trained in the new technologies, which enable them to conduct virtual CCB meetings, on-line polls, and audits).

Many people will need to be trained or retrained on the new tool. Many will have to be trained on the concepts of SCM and its functions. In the author's experience, it makes a tremendous difference if people are first trained regarding the concepts of SCM and then trained on the tool. In this way, they can correlate what they are doing with the tool with the actual SCM concepts and functions. If employees are just trained to use the tool without understanding the underlying SCM concepts, then they will be doing their tasks without knowing how their actions affect others. Yes, it is possible to implement an SCM tool and use it without actually knowing anything about the SCM concepts, but in the long run it is better to do it the right way—that is, to provide training in the tools along with training on the SCM concepts and how the concepts are implemented in the tool.

The following issues need to be addressed before the SCM implementation or changeover begins: process, culture, roles, risk management, environment, application, requirements, management, and planning [8].

The organization should address the current process and the new SCM process and how the new tool will be different. The work environment and work culture

of the organization and whether they have to be changed or adapted to meet the needs of the new tools need to be analyzed. This is important because a new tool will bring about changes in the roles of users and give them more power, freedom, and responsibility. The organization should be geared to handle this change.

The new roles of staff need to be clearly defined. Because most processes are automated by the introduction of an SCM tool, many existing jobs will no longer be needed, and many job profiles will change. The implementation team should make the staff aware of the postimplementation scenario and what exactly will happen to their jobs after the tool is implemented. This is important to secure the cooperation of the users, which is a critical success factor for the tool.

The organization should also identify and assess the potential risk factors in making the transition to the new tool and try to reduce or resolve them. In addition, the tool implementation team should consider which projects are going to use the tool and which project will be the pilot project. If the tool is implemented for a single project and not for the entire organization, then the team should answer such questions as which subsystems of the project are to be under control of the tool, whether the work given to the subcontractors will be put under the control of the tool, and which module is going to be chosen for the pilot implementation.

The organization should also have a realistic knowledge of what they can expect from the tool and what its limitations are. This information should also be given to the users of the tool, because overexpectations about a tool can turn into dissatisfaction, misuse, lack of use, or noncooperation when the tool fails to deliver what the user expects it to deliver. Accordingly, users should be educated about the capabilities and limitations of the tool. During and after the implementation, management support is essential for the success of the tool. So management must take active interest in the tool and should designate one of the organization's top executives (with the necessary authority and firepower) as the leader of the tool implementation team. As Dart [8] says, many brave decisions need to be taken, resources have to be used, and schedules have to be altered, and for this one needs a senior person at the helm of the implementation team. Finally, all of these issues should be documented, and components such as plans, cost estimates, budgets, and time schedules should be prepared before the implementation begins.

Once the planning stage is over, the implementation team in association with the vendor's representative can start the implementation. The tool can be first implemented on the pilot project. Selecting the pilot project is another critical factor, because failure in the pilot project can end the implementation process. It is necessary to carefully choose the pilot project, considering the project members, the project environment, and other variables. The users in the pilot project have to be given thorough training in the tool and the SCM concepts and in such areas as how it is going to affect the work environment and how the processes are going to be automated.

Choosing a high-profile project for the implementation is advantageous and at the same time dangerous. It is advantageous in the sense that if the pilot project is an unqualified success, then winning over company-wide acceptance is easy. It is dangerous, because if it fails, everybody will know about it. Accordingly, if the implementation team is sure about making the pilot implementation a success, it is better to choose a high-profile project. If the tools have been chosen correctly, if

the implementation is well-planned, and if the project team is well-prepared and well-trained, then there is no reason why the project should fail. Also, with constant monitoring during the initial phases, any signs of a disaster can easily be detected and corrected. The pilot project will also give information about the organization and its peculiarities that will be very useful when the company-wide implementation is done.

Any organization that is going to implement an SCM tool or change over to a new SCM tool will benefit greatly by reading the Dart's technical paper [8] and Chapter 5 in [9].

Finally, please consider this caveat: The most critical factor determining the success of any SCM tool implementation is the support of the people who use the system. Even the best tools will fail if there is no user support. So the decision of the committee should be a consensus decision. If some people's views are overridden by majority vote, then management should make every effort to make them understand the reasons for the decisions and should spare no effort to win them over. Disagreements are common in any group discussion, but the success of the group lies in the fact that the decisions made by the group are owned by all members of the group, that everybody emerges as a winner, and that the choice was made by the group as a whole. This feeling is very important, because the company will need everyone's goodwill and support to achieve success during implementation and after implementation.

SCM Tools: Make or Buy?

So far we have seen three possible scenarios for implementing an SCM system:

1. The manual system;
2. The semi-automated system, in which some components like the change management system or the build management system are automated;
3. The integrated system, where the configuration management tools are integrated into the development process.

Except for the first case, these implementation scenarios use some sort of SCM tool. The question is whether to make the SCM tools in-house or to buy them.

Why can't companies develop their own SCM tools? Developing an SCM tool is a very complex and time-consuming process that requires a lot of skilled manpower and other resources. Many companies have personnel on their payrolls who can absorb the necessary knowledge and who have experience in developing sophisticated systems. The problem is that SCM tool development is not the main business of these companies. Instead, they should be directing all of their available resources into improving their own products or services so that they can remain competitive and better serve their customers and continue to grow.

SCM tool vendors are people who have invested huge amounts of time and effort in research and development to create packaged solutions. SCM tool vendors spend billions of dollars in research and come up with innovations that make the packages

more efficient, flexible, and easy to implement and use. Also, with the evolution of new technologies, vendors will be able to upgrade their products constantly to take advantage of the best and latest advancements in technology, because their main focus is on improving the capabilities of their tools.

Since designing and implementing SCM tools is not the business of most companies, or a focus of their executives, the systems developed by an in-house team will never equal in quality, scope, functionality, or technology those created by software firms. These software firms (SCM tool vendors) can produce sophisticated packages and provide their clients with products that allow them to maintain a focus on their own chief activities, thus improving revenues, profits, and shareholder returns. During the 1960s, 1970s, and 1980s, there were not many vendors that were producing SCM tools; as a result, companies needing a specialized SCM tool often needed to rely on their own devices and ingenuity to develop something that would suit their needs. In today's marketplace, there are many SCM tool vendors that are actively pursuing all the niches and circumstances where SCM tools can solve problems for development organizations.

However, situations do exist in which a company will have to develop or make its own SCM tools. The main reasons remain factors such as the nonavailability of tools suited for the company or the peculiar nature of the company or the project. For example, an organization the author worked with had more than 2,500 professionals spread around the globe in more than 54 offices in more than 15 countries. The main development was carried out in the headquarters with dedicated lines connecting the client sites in different continents. The professionals were constantly moving from one project to another and from one country to another. The company had a central SCM team and individual teams within each project. The company personnel were connected via e-mail. The main problem the SCM team of the company faced was finding the appropriate people to deal with a CR evaluation, problem evaluation, or auditing. Because employees were always on the move, a person who was on a project today could be on another continent the next day. So the configuration team was finding it difficult to allocate the tasks to the right people. The solution was to create a skill inventory database of all the employees (as described in Chapter 14) that was always kept current and up-to-date. The information was stored in a relational database. In addition, the company had something called a manpower allocation task committee (MATC) that assigned the various professionals to the various locations and projects. The MATC records were always current and updated because MATC did the allocation and coordinated the travel plans. Accordingly, these details (the availability details) were imported from the MATC database to the CM database.

The company also had a good performance evaluation system, which included asking the employees to update their skill inventory—that is, how many years of experience they had in each of their skills. The skills included programming languages, DBMS, GUI design, testing, and auditing. For each skill for which the employee had more than six months of experience, he or she was asked to take an on-line test. The test scores were multiplied by the experience in months to arrive at a point rating system. So the configuration database had the employee availability, the skill set, the competency level, and other statistics relevant to employees such as

their work phone number and e-mail ID. The configuration team could query this database and obtain details about people who were available currently for a task. Because the contact information was also available in the database, the configuration team could contact the concerned person immediately and assign the task. This was a requirement that no tool supported, so the company had to make the tool in-house. The company already had tools for most of the other SCM functions like change management and system building, and this employee tracking tool integrated seamlessly with the other tools and proved very effective in eliminating delays in such processes as CR processing, PR evaluation, and auditing.

In another company, the problem was quite different. This organization had a workflow automation system based on Lotus Notes. All the users in the company were very familiar with the Lotus Notes environment. Bringing in an SCM tool was discussed but discarded, because it would not integrate well with the existing infrastructure. The author was in charge of the SCM implementation. The SCM implementation team discussed the various options and found that the most cost-effective solution was to build a tool in-house. Lotus Notes's inherent strengths in workflow automation, security administration, and Web features made it easier to design and develop the SCM tool. The SCM tool that was finally developed integrated seamlessly with the existing environment and was a huge success. Also, there was no need for extensive training; the project members needed training only in the concepts of SCM and how it was implemented in the system. Because all of them were comfortable with the environment, the transition was almost painless.

To conclude, it is always better to buy SCM tools. Many tools are available free of cost, but the main problem with them is lack of technical support. Also, these tools will not be updated to take advantage of the latest technological developments. Meanwhile, however, the commercial tools are getting better and bigger and have more features. Moreover, they have been developed by people who specialize in developing those types of tools, most of which can be customized to suit your needs. Accordingly, unless, and until, your project or organization has a need that cannot be fulfilled by the available tools, it is better to buy the tools rather than make them.

Summary

This chapter discusses SCM tools and their selection. Except for very small, single-person projects, SCM tools can dramatically improve development productivity. This chapter also discusses how to choose a tool that is right for an organization and how to deploy the tool in that organization. Finally, the chapter examines the decision about whether to make or buy SCM tools.

References

[1] Whiftgift, D., *Methods and Tools for Software Configuration Management*, Chichester: England, John Wiley & Sons, 1991.

[2] Mosley, V., et al., "Software Configuration Management Tools: Getting Bigger, Better, and Bolder," *Crosstalk: The Journal of Defense Software Engineering*, Vol. 9, No. l, Jan. 1996, pp. 6–10.

[3] Weatherall, B., "A Day in the Life of a PVCS Road Warrior: Want to Get PVCS Organized Quickly in a Mixed-Platform Environment?" Technical Paper, Synergex International Corporation, 1997.

[4] Jasthi, S., "SCM Without Tears, "http://pw2.netcom.com/-siasthi/index.htm], 1997.

[5] Dart, S., "Concepts in Configuration Management Systems," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1994.

[6] Dart, S., "Not All Tools are Created Equal," *Application Development Trends*, Vol. 3, No. 9, 1996, pp. 45–48.

[7] Dart S., "Achieving the Best Possible Configuration Management Solution," *Crosstalk: The Journal of Defense Software Engineering*, September 1996, pp. 9–13.

[8] Dart S., "Adopting An Automated Configuration Management Solution," Technical Paper, STC'94 (Software Technology Center), Utah, April 12, 1994.

[9] Dart, S., *Configuration Management: The Missing Link in Web Engineering*, Norwood, MA: Artech House, 2000.

Selected Bibliography

"Change Management for Software Development," Continuus Software Corporation, 1998.

"Cost Justifying Software Configuration Management," PVCS Series for Configuration Management White Paper, Intersolv, 1998.

"Software Configuration Management for Client/Server Development Environments: An Architecture Guide," White Paper, Intersolv, 1998.

"Software Configuration Management: A Primer for Development Teams and Managers," White Paper, Intersolv, 1997.

Alain Abran, A., and Moore, J. W. (eds.), *SWEBOK: Guide to the Software Engineering Body of Knowledge (Trial Version)*, Los Alamitos, California: IEEE Computer Society, 2001.

Alder, P. S., and A. Shenhar, "Adapting Your Technological Base: The Organizational Challenge," *Sloan Management Review*, Fall 1990, pp. 25–37.

Berlack, R. H., *Software Configuration Management*, New York: John Wiley & Sons, Inc., 1992.

Bochenski, B., "Managing It All: Good Management Boosts C/S Success," *Software Magazine* (Client/Server Computing special edition), Nov. 1993, p. 98.

Bones, M., "Technology Audit: True Software Suite," White Paper, Butler Direct Limited, 1998.

Bouldin, B. M., *Agents of Change: Managing the Introduction of Automated Tools*, Englewood Cliffs, NJ: Yourdon Press, 1989.

Bourque, P., and R. E. Fairley (eds.), *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

Cagan, M., and D. W. Weber, "Task-Based Software Configuration Management: Support for 'Change Sets' in Continuus/CM," Technical Report, Continuus Software Corporation, 1996.

Chris, A. "Why Can't I Buy an SCM Tool?" *Proc. ICSE SCM-4 and SCM-5 Workshops (Selected Papers)*, Berlin, Springer-Verlag, 1995, pp. 278–281.

Dart, S., "Past, Present and Future of CM Systems," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1992.

Dart, S., "Spectrum of Functionality in Configuration Management Systems," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1990.

Dart, S., "To Change or Not to Change," *Application Development Trends*, Vol. 4, No. 6, 1997, pp. 55–57.

Dart, S., and J. Krasnov, "Experiences in Risk Mitigation for Configuration Management," *Proc. 4th SEI Conference on Risk*, Monterey, CA, November, 1995.

Dart, S., *Configuration Management: The Missing Link in Web Engineering*, Norwood, MA: Artech House, 2000.

Feiler, P. H., "Configuration Management Models in Commercial Environments," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1991.

Fichman, R. G., and C. Kemerer, "Adoption of Software Engineering Innovations: The Case of Object Orientation," *Sloan Management Review*, Winter 1993, pp. 7–22.

Hall, E. M., *Managing Risk: Methods for Software Systems Development*, Reading, MA: Addison-Wesley, 1998.

Hurwitz, J., and A. Palmer, "Application Change Management-True Software, Inc.," White Paper, Hurwitz Group, 1997.

Kolvik, S. "Introducing Configuration Management in an Organization," *Proc. ICSE '96 SCM-6 Workshops (Selected Papers)*, Berlin, Springer-Verlag, 1996, pp. 220–230.

Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

Mason, R. P., "Enterprise Application Management in the Age of Distributed Computing: The True Software Approach," White Paper, International Data Corporation, 1998.

Parker, K., "Customization of Commercial CM System to Provide Better Management Mechanisms," *Proc. ICSE SCM-4 and SCM-S Workshops (Selected Papers)*, Berlin: Springer-Verlag, 1995, pp. 289–292.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition*), New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

Weber, D. W., "Change Sets Versus Change Packages: Comparing Implementation of Change-Based SCM," *Proc. 7th Software Configuration Management Conf. (SCM7)*, Boston, MA, May 1997, pp. 25–35.

CHAPTER 16

Document Management and Control (DMC) and Product Data Management (PDM)

Introduction

Software development is an activity that produces a lot of documents. According to Visconti and Cook [1], low-quality, poor, obsolete, or missing documentation is a major contributor to low product quality and high development and maintenance costs. Documentation is the written record of what the software is supposed to do, what it does, how it does it, and how to use it.

Virtually everyone agrees that good documentation is important to the analysis, development, and maintenance phases of the software process and is an important software product. It can be said that to develop high-quality software products, high-quality documentation is a must. Most information and documentation these days is in digital form, and its format is completely different from that of conventional documents. State-of-the-art tools are available to share and convey information as efficiently as possible. All software projects involve the production and control of documentation.

According to Wallace [2], there are mainly four types of documents that are produced by a software development project:

- *External deliverable (permanent):* Permanent documentation as a deliverable from the project (e.g., "help" information, user manuals, and training materials);
- *External deliverable (temporary):* Temporary documentation that is an external deliverable from the project but that has no value once the project has been completed (e.g., discussion papers, draft documents, and interim progress reports);
- *Used by project team (permanent):* Permanent documentation to support the maintenance and enhancement of the system (e.g., design specifications, database definitions, source code, and process diagrams);
- *Used by project team (temporary):* Temporary documentation that is only for internal communication (e.g., ideas, issues, control, and working papers).

The above factors—permanency and target audience—affect the requirements for quality, review, and update. For example, external deliverables need to be of high quality, whereas internal documents may be informal and incomplete; permanent

documentation will need to be updated as circumstances change, but temporary documentation will usually be left unchanged or disposed of after use.

There will be many types of documents with varying purposes, natures, and life cycles. Some documents like standards and guidelines that are used in projects are produced even before the start of the projects, and many of these documents (like standards) are created by other organizations. After the development project has been completed, documents will still be produced during the operation and maintenance phases. Many documents get updated and modified during the course of the project. This will make the older versions obsolete, and their continued usage can create a lot of problems. The large amount of documents makes it more difficult to locate the information required. Thus, it is essential that all the project stakeholders—from clients to end users—use the correct version of the documents. This is an enormous and difficult task—one that requires the institution of systematic and scientific procedures and methods to manage the creation, classification, modification, change, distribution, and archiving or deletion of the documents. The process that ensures that everyone in a project uses the current and correct versions of the documents, that changes made to the documents are effected in all copies of the document, and that unauthorized changes are not made to the documents is called DMC. DMC is an integral part of CM.

The primary purpose of DMC is to provide the right users with the right information at the right time. It should provide an efficient way of sharing knowledge, information, and thinking among the project's participants. All participants should find it easy to consult the project's documentation repository to find all content that is relevant to their interests. It should be equally easy for them to lodge in the repository any documented information that they feel is of value to the team—in a systematic and controlled manner.

Document Life Cycle

Like software products, documents have a life cycle. They also start as an idea or concept, and once they are used satisfactorily, they are either archived or destroyed. DMC activities focus on enabling documents to service their users during the life cycle. The documentation process records information produced by a life cycle process or activity. The process contains the set of activities that plan, design, develop, produce, edit, distribute, and maintain those documents needed by all concerned, including managers, engineers, and users of the system or software product.

The different phases of the document life cycle are creation, storage, publishing, viewing, modification, review, approval, retention, archiving, and disposal. Figure 16.1 shows the document life cycle phases. These phases are not linear; many phases occur more than once during the life cycle. For example, once the document is created, published, and viewed, modifications or changes are made. Once the modifications are made, the document is reviewed by a competent authority. Once the reviewer approves the changes, the documents are again published and viewed. This modification, review, and approval process happens many times during the life cycle of a document. Unauthorized modification and publishing without proper

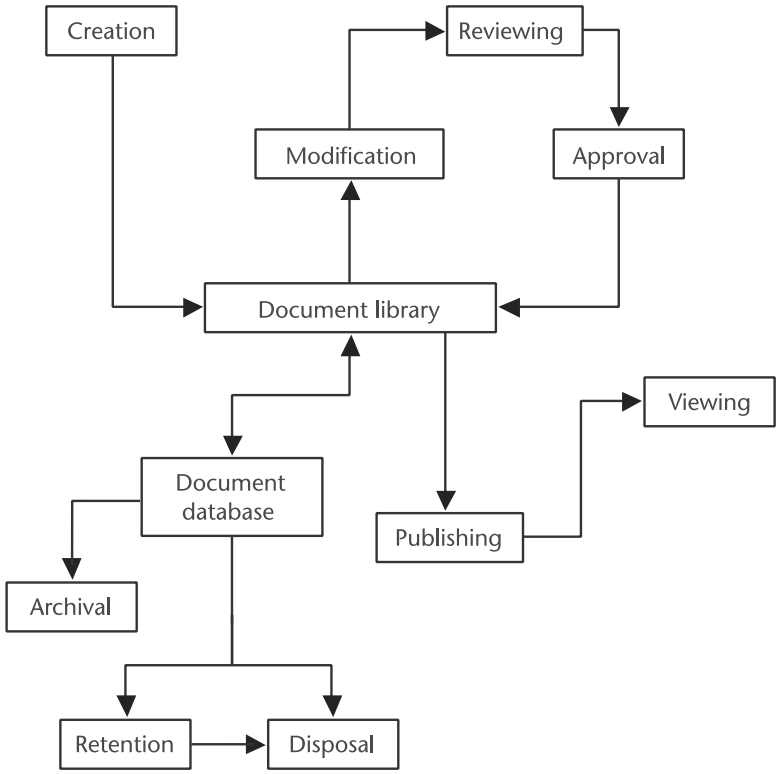


Figure 16.1 Document life cycle.

review and approval can cause problems for the users of the document. Preventing this from happening is the job of a good DMC system. Next, we examine the different phases of the document life cycle in a little detail.

Document Creation

Documents can be created in a number ways and from a number of sources. Documents can be written by people in your organization, external consultants, or partner organizations or can be bought from external sources. Usually, standards, statutory regulations, and guidelines are developed by external organizations. Documents developed by consultants include implementation guidelines and programming standards where the consultant has more expertise than company employees. If your organization works with other organizations or develops products for other organizations, documents like design specifications and interface control documents might come from those organizations. Then, there are documents that are created by your own organization like project plans, SCM plans, requirements specifications, design documents, test plans, and user manuals. These documents, which can come in a variety of formats and on a variety of media using a variety of tools, can be text, images, sound, video, or a combination of all the above. All the documents that are created or are procured from external sources should be subjected to review and approval by authorized personnel before being placed in the document library.

Document Storage

Documents are stored depending on the format. Paper documents need to be stored in filing cabinets, while digital documents need to be stored on a hard disk. The place where the documents are stored is called the document library. The library contains all the active documents that are needed by the project. With the advances in technology and the reduction in the cost of digital storage and retrieval mechanisms, most documents today are stored in digital form on a server that the project participants can access. The document library should have control and safety measures that prevent unauthorized viewing, modification, and deletion.

Publishing

Publishing includes activities related to document presentation. The same content or data could be presented in different formats—such as text, graphs, and video clips. The ability to separate the content of a document from its format is increasing. Accordingly, the same sales data could be presented as a text document, a spreadsheet, or a set of graphs. Each presentation format has its own advantages. Keeping the content and format independent of one another gives the publisher the flexibility to choose the format that is best suited for each situation. Another function of publishing is to combine the different documents or different document elements (e.g., text, pictures, and sound) in a single document and make them accessible to users. Even though the different document elements exist in the document library separately, the process of combining them should be transparent to users.

Viewing

The documents are created and stored so that end users can view the documents. In most cases, users will be employing the same tools used for creating the document for viewing. This could create problems if users do not have the tools with which the document was created. To avoid this problem, the document management uses standard formats like XML, HTML, and PDF. In other words, more and more tool independence is achieved. In this way, presentation tools like Web browsers and PDF readers are not part of the DMC system. This approach has dramatically improved the accessibility of documentation and information sharing.

Modification or Change, Review, and Approval

Once documents are created and published, and as people start using them, the need for modification or making changes arises. The modification could be due to many factors including errors in the documentation, changes in the environment for which the document was created, change in technology, and change in organization's priorities. All changes to documentation should be in writing and processed in a manner that ensures prompt action. All changes will be documented by a formal change procedure and will be approved by the technical authority responsible for originating and revising the document. Once the need for changing the document

has been identified, the document is checked out from the document library, and the changes are made. Any other documents impacted by the changes in the current document are also changed. The modified documents are reviewed and approved by authorized personnel and then returned (check-in) to the document library and given a new version number. This is similar to the change control process described in Chapter 8.

Records Retention

With time, the value of many documents will diminish, and maintaining these documents is a waste of space and effort. Documents fall under three categories—documents that do not have any value after the completion of the project, documents that have value as reference material for future projects, and documents that should be retained to satisfy some statutory requirement or contract regulations.

Documents and records that fall into the last category come under the purview of records retention. The records retention policy will usually be specified in the SCM plan—detailing decisions regarding documents that are to be retained and for how long. Accordingly, the records and documents that are to be retained will not be destroyed until the specified period of time has passed.

Document Disposal

The documents that are not needed after the completion of the project like status reports, internal memos, and draft versions are permanently deleted to free up storage space. Document disposal need not occur only at the end of the project. It can be a periodic activity. Deleting unwanted and obsolete documents from the document library will reduce clutter and will improve the efficiency of document retrieval systems. A word of caution: before any documents are removed or deleted from the project, the organization's records retention policy should be reviewed and complied with.

Archiving

Records and documents that have a sustaining utility exceeding storage costs are preserved permanently in an archive for long-term storage. The difference between records retention and archiving is that retention is performed to carry out some legal or contractual obligation and has a time dimension attached to it, whereas archival work is done because of the value of the document as a reference material, and there is no time limit for how long the archived documents will be kept. Usually, documents are archived to magnetic media or some other backup mechanism.

Documentation and SDLC Phases

We have seen that of the various documents used during the software life cycle, some documents such as standards and guidelines are available much before the

requirements analysis or conceptualization phase. Many documents are created during the development phase; many more documents are created during the operation and maintenance phases. Most documents undergo revisions and modifications during the course of the development of the product. Table 16.1 outlines some typical documents used during the development, operation, and maintenance of a software system.

Table 16.1 Documentation and SDLC Phases

Document name	*Conceptualization*	*Requirements Definition*	*Systems Analysis & Design*	*Coding & Unit Testing*	*Integration Testing*	*Implementation*	*Operation & Maintenance*	*Retirement*
Standards and guidelines	X	X	X	X	X	X	X	X
Project plan (PP)		X	X	X	X	X	X	X
SCMP		X	X	X	X	X	X	X
RDD		X	X	X	X	X	X	X
SAD			X	X	X	X	X	X
SDD			X	X	X	X	X	X
CRs		X	X	X	X	X	X	
PRs		X	X	X	X	X	X	
Impact analysis report		X	X	X	X	X	X	
SCM documentation		X	X	X	X	X	X	X
CSA reports		X	X	X	X	X	X	
STS			X	X	X	X	X	X
STP			X	X	X	X	X	X
UTS				X	X	X	X	X
UTP				X	X	X	X	X
Audit and review reports		X	X	X	X	X	X	X
Test results				X	X	X	X	X
User documentation					X	X	X	X
Enhancements requests						X	X	X

DMC

DMC is the discipline of creating and managing the documents used in a project. The primary objective of this discipline is to ensure that documents are created, stored, changed, used, and disposed of in a systematic and scientific manner. This system ensures that the right information is available to the right personnel at the right time. It also makes sure that unauthorized access, modification, and corruption of documents does not happen; that any change made to a document is made after formal approval; and that once the changes are made the documents are reviewed and approved before they are reissued for use. The use of the DMC process should achieve the following objectives:

- Identify all documents to be produced by the process or project;
- Specify the content and purpose of all documents and plan and schedule their production;
- Identify the standards to be applied for development of documents;
- Develop and publish all documents in accordance with identified standards and in accordance with nominated plans;
- Maintain all documents in accordance with specified criteria.

ISO 9001 [3] specifies that organizations must establish and maintain documented procedures to control all documents and data including, to the extent applicable, documents of external origin such as standards. Documents and data can be in the form of any type of media, such as hard copy or electronic media.

Documents and data must be reviewed and approved for adequacy by authorized personnel prior to issue. A master list or equivalent document control procedure identifying the current revision status of documents should be established and be readily available to preclude the use of invalid or obsolete documents.

Changes to documents and data should be reviewed and approved by the same functions or organizations that performed the original review and approval, unless specifically designated otherwise. The designated functions or organizations should have access to pertinent background information upon which to base their review and approval. Where practicable, the nature of the change should be identified in the document or the appropriate attachments. The documentation should be traceable, handled, stored, responsive to change requests, and consistent with the SCM plan.

The organization must establish and maintain procedures for controlling all documents to ensure the following [4]:

- That they can be found;
- That relevant copies of appropriate documents are available at all locations where operations essential to the effective functioning of the quality system are performed;
- That they are periodically reviewed, revised as necessary, and approved for adequacy by authorized personnel;
- That current versions of relevant documents are available at all locations where operations essential to the effective functioning of the environmental management system are performed.

- That obsolete documents are promptly removed from all points of issue and points of use, or otherwise assured against unintended use;
- That any obsolete documents retained for legal or knowledge preservation purposes are suitably identified.

Documentation must be legible, dated (with dates of revision) and readily identifiable, maintained in an orderly manner, and retained for a specified period. Procedures and responsibilities should be established and maintained concerning the creation and modification of the various types of document.

According to IEEE/EIA 12207.0 [5], every document should have the following parts:

- Title or name
- Scope and purpose;
- Intended audience;
- Procedures and responsibilities for inputs, development, review, modification, approval, production, storage, distribution, maintenance, and configuration management;
- Schedule for intermediate and final versions.

In addition, documents should have the following:

- Date of creation;
- Copyright information;
- Identification (an identification scheme should be developed that will be descriptive and give an indication of the date of creation, revision, and name of the document);
- Created by (name of the organization, department, and person);
- Distribution list (list of people to whom the copies of the documents are distributed);
- Change history [list of changes (additions, deletions, or modifications) with date of change and signature of the persons who authorized, reviewed, and approved the change];
- Security classification (indicating whether the document is for the general public or classified and restricted for certain users);
- Suitable identification marks (line or symbols) indicating the places where changes have been made—in the case of revised documents;
- Table of contents;
- Glossary of terms used in the document;
- Errata (if any).

According to ISO 10007 [6], to protect the integrity of the configuration and to provide a basis for the control of change, it is essential that CIs, their constituent parts, and their documentation be held in an environment meeting the following conditions:

- It is commensurate with the environmental conditions required (e.g., for computer hardware, software, data, documents, or drawings).

- It protects them from unauthorized change or corruption.
- It provides means for disaster recovery.
- In the case of software, data, documentation, and drawings, it permits the controlled retrieval of a copy of the controlled master.
- It supports the achievement of consistency between the as-built or produced state of a configuration and the as-designed state.

Operation of the DMC system during the project should be made as easy and efficient as possible, but without losing the degree of control and audit that is required. Usually, the task of DMC is performed by the SCM function. Individual participants should be able to access and request changes and enhancements and point of errors to the documents. There should be well-defined procedures, authorities, and controls for approving the CRs, checking out, making the changes, reviewing, approving, checking in, and promoting the documents.

The DMC system should also track changes to documents that come from external sources like standards and new versions or updates of the standards, and other external documents should be incorporated whenever necessary. For example, consider a situation where a revised version of a standard that is used in the project has been released. This standard was used for creating some other documents (e.g., a QA manual) used by the project. In such cases, the new standard should be brought under the document control, all the documents affected by the revision to the standard should be identified, and the necessary modifications should be made.

The documentation manager should know the status of each document and should have a reporting mechanism (similar to the CSA) to know such information as which documents are under creation and their estimated date of completion, how many documents are under review, how many people are using a particular document, and whether the obsolete versions should be deleted or archived.

At the end of a phase or stage of the project, all planned deliverables should have been completed, finalized, approved, and distributed. Internal documents should also have been completed, subjected to the defined reviews, and finalized. Quality audits should be conducted to ensure that all planned items had been produced in accordance with the defined controls and procedures.

At the end of the project, permanent items will be retained for future use (for example, for the maintenance and support phase), and external deliverables will be distributed to the customers or end users, and master copies will be maintained.

PDM and DMC

Management of product data is an activity that exists in the manufacturing industry. Product data—text and drawings—were traditionally archived on paper. However, advancements in technology and the proliferation of computers have greatly reduced the need for paper documents, and today product data is stored in a digital format. The business environment is becoming highly competitive as companies have to compete on a global basis. To compete in today's highly competitive marketplace, manufacturers must find new ways to reduce cost, improve quality, improve developmental productivity, and reduce the time to market. PDM is a discipline that

allows organizations to become more competitive by better managing and controlling their product data.

Software is becoming more and more important, and almost all systems from toy planes to fighter planes are controlled by software systems. Both software professionals and hardware engineers have to work hand-in-hand to create the products that are required today. This collaboration necessitates that professionals from both disciplines learn about how the other side is performing its activities. Accordingly, today's software professionals should learn the fundamentals of PDM to be more competitive and help their counterparts who are engaged in the design and production of hardware in a better and more efficient manner.

Many of the functions that are performed by PDM are closely related to DMC. In both PDM and DMC, the activities performed to manage the product or project data are the same or similar. While PDM keeps track of the data and documents related to the hardware artifacts, DMC does the same for the software artifacts. Thus, it is beneficial and necessary to be aware of the close relationship that these two disciplines share.

Overview of PDM

PDM is the discipline of controlling the evolution of a product and providing other procedures and tools with accurate product information at the right time in the right format during the entire product life cycle [7]. So PDM involves supporting departments such as manufacturing, marketing, sales, and purchasing. It also involves supporting the organizations vendors, subcontractors, and business partners. PDM is also known as collaborative PDM (cPDM), product information management (PIM), PLC management (PLCM), and electronic PDM (ePDM), among other monikers. PDM performs the following functions to achieve its objectives:

- *Data vault management:* Every PDM system has a centralized data vault where all the master copies of all the documents, drawings, and other data is stored. The data vaults can be filing cabinets, secured directories in a computer system, or even databases. The data stored in these data vaults can come from various applications such as data from computer-aided design (CAD), computer-aided engineering (CAE), computer-integrated manufacturing (CIM), operating instructions, service manuals, drawings, technical specifications, and reports. In addition to this data, the PDM system stores information about the various documents (called metadata) like the title, date of creation, names of the persons created, review and approval of the document, and history of changes made. The metadata is used to search the database for a particular document or information.
- *Workflow management:* During the development of a product, many thousands of parts may need to be designed. For each part, files need to be created, modified, viewed, checked, and approved by many different people, perhaps several times over. What is more, each part will call for different development techniques and different types of data—solid models for some, circuit diagrams for others, and finite element analysis (FEA) data for others. As

if this is not confusing enough, work on any of these master files will have a potential impact on other related files. As a result, there needs to be continuous cross-checking, modification, resubmission, and rechecking. With all these overlapping changes, it is all too easy for an engineer in one discipline to invest considerable time and effort in pursuing a design that has already been invalidated by the work someone else has done in another part of the project. Bringing order to this highly complex workflow is what PDM systems do best. In particular, they keep track of the thousands of individual decisions that determine who does what next.

- *Product structure management:* Every product is made up of thousands of component parts. A product can have many subassemblies, each of which can be made of many more subassemblies. A product structure is a division of parts into a hierarchy of assemblies and components until it reaches the lowest level—the component level. The product structure is comprised of components and the properties of the components and the relationship between them. The major activities performed by product structure management are identification and control of product configurations, linking product data to the structure, and transferring the product structure and other data between PDM and material resource planning (MRP) and enterprise resource planning (ERP) systems.
- *Classification management:* This function deals with the classification of standard components in a uniform way. The main objective of classification is to promote the reuse of components. The components are classified and information (attributes) about them is stored in the PDM database, in such a way that promotes reuse. The designers can search the PDM database for components that are suited for their purpose, and only if such a component does not exist, they need to think of a new component. By improving reusability, the number of components in a product can be reduced, which can result in shorter design times, fewer parts to purchase, and reduced time to market.
- *Communication and notification:* Workflow management supports automatic notification and communication. The system can be programmed to automatically send notification when specified events occur (such as when an item has been checked in to the database or a change has been approved). Users can be given the authority to subscribe to the notification that is of interest to them. This automatic notification improves the overall awareness of the project team members about what is happening in the project.
- *Query management:* As you can imagine, you need to be able to "get at" the components and assembly data by a variety of routes. You can move up and down a classification tree, pick your way through a product structure, simply call up the data you want by searching for it by name or part number, or search for groups of data by specifying an attribute or combination of attributes. The query management function allows you to do this using querying languages like SQL or graphical interfaces.

In today's highly competitive business environment, the challenge is to maximize the time-to-market benefits of concurrent engineering while maintaining control of your data and distributing it automatically to the people who need it, when

they need it. PDM systems cope with this challenge by holding the master data in a centralized secure vault where its integrity can be assured and all changes to it monitored, controlled, and recorded.

Data Management

Manufacturing companies are usually good at systematically recording component and assembly drawings but often do not keep comprehensive records of attributes such as "size," "weight," and "where used." As a result, engineers often have problems accessing the information they need. This leaves an unfortunate gap in their ability to manage their product data effectively. PDM systems should be able to manage attribute and documentary product data, as well as relationships between them, through a relational database system. In the world of information explosion where too much data is being generated, a technique to classify this information easily and quickly needs to be established.

Classification is a fundamental capability of a PDM system. Information of similar types should be capable of being grouped together in named classes. More detailed classification would be possible by using "attributes" to describe the essential characteristics of each component in a given class.

Process Management

Data management, which includes organizing data so that it is easy to access, refer to, and cross-reference, is comprised of passive procedures. Process management, on the other hand, is about controlling the way people create and modify data—active procedures. Process management systems normally have three broad functions:

1. They manage what happens to the data when someone works on it.
2. They manage the flow of data between people.
3. They keep track of all the events and movements that happen in functions 1 and 2 during the history of a project.

Engineers create and change data for a living. The act of designing something is exactly that. A solid model, for example, may go through hundreds of design changes during the course of development, each involving far-reaching modifications to the underlying engineering data. Often, engineers will wish simply to explore a particular approach, later abandoning it in favor of a previous version. A PDM system offers a solution by acting as the engineer's working environment, meticulously capturing all new and changed data as it is generated, maintaining a record of which version it is, recalling it on demand, and effectively keeping track of the engineer's every move. Thus, when engineers are asked to carry out a design modification they will have all the information they need in the PDM system. Since the PDM system stores the documents and drawing in a central database, the access to which is controlled, the chances of two people working on the same component is eliminated. Also, once the modifications are complete and the item is checked in,

the automatic notification feature will inform all the concerned parties about the change, thus preventing the production or purchase of obsolete and unwanted items.

PDM systems should not just keep comprehensive database records of the current state of the project, they should also record the states the project has been through. This means that they are a potentially valuable source of audit trial data. The ability to perform regular process audits is a fundamental requirement for conformance to international quality management standards such as ISO 9000, EN 29000, and BS 5750. However, project history management is also important to allow you to backtrack to specific points in a project's development where a problem arose, or from which you may wish to now start a new line of development. The PDM system captures the changes and modifications to the components and the system. This creates a work history for the product development. This level of historical tracking, as well as providing comprehensive auditing, also permits the active monitoring of individual performance.

Benefits of PDM

Some of the benefits of PDM systems are reduced time to market; improved design productivity, creativity, and accuracy; better control of the project; and better engineering change management. Properly managed data will help productivity significantly even when a single workgroup is using it. However, PDM achieves its real potential in an enterprise-wide environment, supporting and coordinating the activities of many teams. The benefits result from managing data throughout product life; sharing of data or information among users; supporting concurrent processes; and reducing product cost through increased reusability, development productivity, and accuracy. The following sections discuss the benefits of PDM.

Reduced Time to Market

This is the major benefit of a PDM system. Three factors serve to place limits on the speed with which you can bring a product to market. One is the time it takes to perform tasks, such as engineering design and tooling. Another is the time wasted between tasks, as when a released design sits in a production engineer's inbox waiting its turn to be dealt with. The third is time lost in rework. A PDM system can do much to reduce all these time limitations:

- It speeds up tasks by making data instantly available as it is needed.
- It supports concurrent task management.
- It allows authorized team members access to all relevant data, all the time, with the assurance that it is always the latest version.

Improved Design Productivity

With a PDM system providing the engineers with the correct tools to access data efficiently, the design process itself can be dramatically shortened. By reducing the time engineers need to spend searching for the right information and tools and

allowing them to concentrate on designing, an efficient PDM system gives engineers more time to design.

Improved Design and Manufacturing Accuracy

An important benefit of PDM systems is that everyone involved in a project is operating on the same set of data, which is always up to date. This eliminates overlapping or inconsistent designs even when people are operating concurrently. Naturally, this leads to far fewer instances of design problems that only emerge at manufacturing.

Better Use of Creative Team Skills

Designers are often conservative in their approach to problem solving for no other reason than the time penalties for exploring alternative solutions are so high. The risks of spending excessive time on a radically new design approach that may not work would be unacceptable. PDM opens up the creative process in three important ways. First, it keeps track of all the documents and test results relating to a given product change, minimizing design rework and potential design mistakes. Second, it reduces the risk of failure by sharing the risk with others and by making the data available to the right people fast. Third, it encourages team problem solving by allowing individuals to bounce ideas off each other using the packet-transfer facility, knowing that all of them are looking at the same problem.

Data Integrity Safeguarded

The single central vault concept ensures that, while data is immediately accessible to those who need it, all master documents and records of historical change remain absolutely accurate and secure.

Better Control of Projects

PDM systems enable you to retain control of projects by ensuring that the data on which they are based is firmly controlled. Product structure, change management, configuration control, and traceability are key benefits. Control can also be enhanced by automatic data release and electronic sign-off procedures. As a result, it is impossible for a scheduled task to be ignored, buried, or forgotten.

A Major Step Toward Total Quality Management

By introducing a coherent set of audited processes to the product development cycle, a PDM system should go a long way toward establishing an environment for ISO 9000 compliance and total quality management (TQM). Many of the fundamental principals of TQM, such as “empowerment of the individual” to identify and solve problems are inherent in the PDM structure. The formal controls, checks, change management processes, and defined responsibilities should also ensure that the

PDM system you select contributes to your conformance with international quality standards.

PDM and SCM

PDM is a class of enterprise software that manages product data and relationships—facilitating innovation and increasing engineering productivity. It allows you to manage, control, and access data surrounding new product design, engineering, and manufacturing processes. By providing controlled and secure global data access, PDM empowers organizations to deliver higher quality products to market faster and more efficiently. This process impacts the entire life cycle of a product, as employees at each phase in the product development process can access the right information at the right time.

SCM, as discussed previously, is the discipline of identifying the configuration of a system at discrete points in time for the purposes of systematically controlling changes to this configuration and maintaining the integrity and traceability of this configuration throughout the system life cycle. SCM is a collection of techniques that coordinate and control the construction of a software system. Today's software systems consist of a myriad of component parts, each of which evolves as it is developed and maintained. SCM ensures that this evolution is efficient and controlled, so that the individual components fit together to form a coherent whole.

Accordingly, we can say that while PDM ensures that product development proceeds smoothly, SCM makes sure that software development is done efficiently. Today most products—both hardware and software—are becoming more and more complex. Another fact is that in today's environment neither software nor hardware can exist in isolation. Software is the integral part and the driving force in almost all machines and systems from mission-critical applications like controlling the operations of satellites and intercontinental ballistic missiles (ICBMs), air traffic control (ATC) systems, managing the functioning of banks and hospitals, and handling the airline and railway reservation systems to performing mundane tasks like operating a door locking system.

In the past the SCM and PDM disciplines have existed and evolved with little interaction. The SCM and PDM disciplines can help an organization to achieve greater efficiencies in its product development, marketing, and customer support efforts. However, the days of SCM and PDM working in isolation are gone. To survive, thrive, and successfully compete in today's highly competitive business environment, organizations must conceive, build, test, and market high-quality products in the most efficient and effective manner. Jobs such as customer support, bug fixes, product enhancements, and product evolution must be done quickly (at Internet speed) and with minimum wasted effort. To achieve these goals, the SCM and PDM disciplines must be integrated.

Only when the hardware and software development teams fully understand what is happening in the "other world," only when the hardware-software boundaries disappear, and only when seamless information integration occurs between the hardware and software development environments will organizations be able

to realize their full potential and become market leaders. If this is to happen, organizations should integrate their SCM and PDM efforts.

PDM Resources

There are many sources for learning more about PDM. However, software professionals need to be familiar with PDM and how it relates to SCM. The best book toward that effort is the following:

- Crnkovic, I., U. Asklund, and A. P. Dahlqvist, *Implementing and Integrating Product Data Management and SCM*, Norwood MA: Artech House, 2003.

Additional resources for comprehensive information about PDM are listed as follows:

- Antti Saaksvuori, A., and A. Immonen, *Product Lifecycle Management*, Berlin: Springer Verlag, 2003.
- Belliveau, P., *The PDMA Tool Book for New Product Development*, New York: John Wiley & Sons, 2002.
- Burden, R., *PDM: Product Data Management*, Eau Claire, WI: Resource Publishing, 2003. PDM Information Center (http://www.pdmic.com/).
- Rosenau, M. D., et al., *The PDMA Handbook of New Product Development*, New York: John Wiley & Sons, 1996.
- Workflow Management Coalition (http://www.wfmc.org/).

Summary

Low-quality, poor, obsolete, or missing documentation is a major contributor to low product quality and high development and maintenance costs. DMC helps to produce high-quality software by managing and controlling—in a scientific and systematic manner and using well-defined procedures and controls—documents through all phases of a product's life cycle. Accordingly, this chapter reviews the requirements specified by the various standards for the proper functioning of the DMC process.

PDM is the discipline of controlling the evolution of a product and providing other procedures and tools with accurate product information at the right time in the right format during the entire product life cycle. PDM and DMC are closely related and perform similar functions to achieve their goals. SCM is a collection of techniques that coordinate and control the construction of a software system. Because today's products rely on tightly integrated hardware and software components, system and software engineers and project and product managers need to have an understanding of both PDM and SCM.

References

[1] Visconti, M., and C. Cook, "Software System Documentation Process Maturity Model," *Proceedings of the 1993 ACM conference on Computer science*, 1993, pp. 352–357.

[2] Wallace, S., The ePMBook, http://www.epmbook.com/, 2004.

[3] ISO, *Quality systems—Model for Quality Assurance in Design, Development, Production, Installation and Servicing (ISO-9001: 2000)*, Geneva, Switzerland: ISO, 2000.

[4] ISO, *Environmental Management Systems—Specification with Guidance for Use (ISO-14001: 1996)*, Geneva, Switzerland: ISO, 1996.

[5] IEEE, *Industry Implementation of International Standard ISO/IEC 12207: 1995, (ISO/IEC 12207) Standard for Information Technology—Software Life Cycle Processes (IEEE/EIA 12207.0-1996)*, New York: IEEE, 1998.

[6] ISO, *Quality management—Guidelines for Configuration Management [ISO-10007: 1995 (E)]*, Geneva, Switzerland: ISO, 1995.

[7] Crnkovic, I., U. Asklund, and A. P. Dahlqvist, *Implementing and Integrating Product Data Management and SCM*, Norwood, MA: Artech House, 2003.

Selected Bibliography

Bourque, P., and R. E. Fairley (eds.), *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

IEEE, *IEEE Standard for Software User Documentation (IEEE Std-1063-2001)*, New York: IEEE, 2001.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

Wallace, S., The ePMBook, http://www.epmbook.com/, 2004.

Watts, F. B., *Engineering Documentation Control Handbook (2nd Edition)*, Norwich, NY: Noyes Publications/William Andrew Publishing LLC, 2000.

CHAPTER 17

SCM Implementation

Introduction

Implementing an SCM system in an organization is not an easy task, because the implementation needs support from a lot of departments, from top management to the developer or programmer. It probably involves changing the way the organization is doing software development and maintenance. The SCM system will introduce new procedures and controls, and people will have to follow those procedures to get something done.

In a non-SCM scenario, changes could be made at will, but once SCM is implemented change management procedures have to be followed. SCM also brings accountability, because all events are recorded and these records could be used to trace the person who made a particular change or modified a program's source code. Thus, resistance to SCM may be a factor that can create many a problem during an SCM implementation.

As with any project, SCM implementation also cannot succeed without the complete cooperation of all people involved, whether it is the top managers who will use the status accounting reports to monitor the project or the developer who has to follow the SCM procedures. This chapter examines some of the techniques that will help overcome resistance to SCM and any other problems during an SCM implementation.

Managing the Implementation

The nature of the SCM implementation is such that it is best handled within a project management context. The implementation involves a series of activities that do not fit naturally into the normal business cycle of events. These activities are of finite duration. They have an endpoint, after which any further work should become absorbed into the normal business activity. SCM also requires allocated resources and the development of specific skills. It is multidisciplinary and team-oriented. The complexity is compounded by the need to involve an increasing number of people over time, distracting them from their normal activities. Thus, this way of working tends to contrast with the software development process businesses.

For project management to be effective, it needs to have the right environment. One of the main reasons for a failed project can be the wrong environment. This can take many forms, including a counterproductive organizational culture or inadequate corporate commitment or sponsorship.

Preimplementation Tasks—Getting Ready

There are at least two truisms that fit most SCM projects: Implementation will be painful, and financial benefits realized in the first year will be minimal. Preparing your company for implementation is almost as important as the project itself. Too often, in justifying a project, we downplay the risks and pain of implementation, afraid that the truth will mean disapproval. However, brutal honesty at the onset is always better than giving team members false expectations. If employees believe the project will progress painlessly, they will only get discouraged when a problem does occur. If you tell employees the truth, they can anticipate pitfalls and prepare for them.

It is important to let everyone know that after many months of implementation preparation, system operations may not go smoothly, and the pain can last as long as three to four months more even if everything has been done correctly. If the implementation goes well, all the better, but if things go wrong as they sometimes will, everyone will be much more understanding. You can get to the same point with all the same problems and be viewed with either approval or disdain depending on how well you have prepared your company. Often, project promoters claim new systems will impact the bottom line immediately. However, experience shows that many systems benefits do not occur until the second year.

Importance of Preparation

Management too often plunges into SCM being less than fully informed, or worse yet with very limited or no knowledge of what to expect. Often, there is a misconception that the skills necessary to select SCM software and then implement it already exist in the organization. Some skills may exist but rarely to the extent necessary to effectively implement SCM within a reasonable time frame. Enter the consulting "experts" from a systems integration firm (often labeled as management consultants) who helped sell you the system, with a software business partner. Worse, when the implementation "experts" arrive, some of them are so inexperienced that it should make you quickly reassess implementation risk. It is of paramount importance with such a high risk and reward ratio that you be absolutely certain the necessary knowledge and skills are present in your implementation team.

Another commonly overlooked issue and, therefore, one that is not well prepared for, is IT change. Often the IT infrastructure changes required to enable the implementation of a new SCM system are not given the high priority they deserve. Certainly, business issues, rather than those related to technology, should drive the implementation of SCM. However, it is the understanding and skills of the IT personnel that supports the technology that enables the business process issues to be improved. Failing to consider that new IT is going to require preparation through education, in order to understand the new technology, is asking for trouble. For example, the IT staff of the organization is very knowledgeable about COBOL and HP-UX, but its new SCM will be using very different technology. IT personnel's understanding of the new technology is an absolute requirement for a successful

SCM implementation. Furthermore, the IT personnel have to make the technology transition fast. If the necessary technology adoption and infrastructure transition are not done well, it will at the very least, delay the project.

One of the biggest problems that many have had with implementing SCM is misunderstanding what SCM is all about and underestimating what it takes to effectively implement. Driving SCM preparation and implementation, senior operating management cannot relegate critical decisions to workers who may not have the background or temperament for this type of decision-making.

Companies need a well thought out and comprehensive process that will help plan, guide, and control the entire SCM implementation effort. Starting an implementation with an undocumented, skimpy, or untailored implementation methodology is an open invitation to disaster and at the very least, a long, drawn-out implementation. Everyone from the boardroom to the stockroom needs to clearly understand their roles and responsibilities for implementation and above all, encourage dialogue that will get people focused on the business objectives as well as early identification or correction of any problems. In addition—and no small matter—the questions of how, when, and who will be accountable for results must be an integral part of this understanding. An implementation that is going astray becomes recognizable when repeated schedule slippages surface. As time moves on, the schedule, missed problems starts to compound the implementation quality, as the invariable response is to start taking shortcuts and bypassing critical business issues.

Before You Leap

An SCM system is a very powerful computer tool, and organizations can gain a lot of competitive advantage by implementing one. Unfortunately, many SCM projects have not been effective and hence unable to achieve the results envisaged. As the cost of an SCM implementation project is very high, it is critical for an organization to make the project a success and start deriving benefits from it as fast as possible. The characteristics of a successful SCM implementation project are described as follows:

- A well-defined project organization structure that details the project planning, execution, and monitoring mechanism;
- An attitude that stresses business transformation instead of process automation;
- An approach that brings about the proper integration of people, processes, and technology through effective management of change;

Companies need a well thought out, comprehensive process to help plan, guide, and control the entire SCM implementation effort. Before the nitty-gritty of software selection begins, management should know how current strategy, processes, and supporting systems work and whether any changes should be made before the new information system is introduced. The preimplementation stage is the time to reconsider the way you do business and to make a detailed analysis of the requirements and the expectations of the new information system. Optimization of software development processes rather than technical innovation should be the focus of an

SCM implementation project. Start defining software needs by examining current processes that govern your flow of information and material throughout the order-to-delivery process and ultimately the entire supply chain. There is a common tendency to shortcut this important activity, but you will pay—sometimes dearly—in time and money for avoiding this essential step.

Take the time to evaluate your SCM plan before you commit to software acquisition and installation. Doing it right the first time is the only way to go. There are many people out there who wish they had taken a brief pause to evaluate their direction. The following questions do not cover every possible contingency but they should be used to stimulate thought and discussion and the right action:

- How do we want to run our business?
- What problems need to be resolved?
- Do we know and understand our priorities?
- Do we fully understand our "as-is" condition versus our "could-be" process?
- Have we defined an action plan for preimplementation preparation activities?
- What tasks will be accomplished and by what dates?
- What are the "missing links" in our software of choice?
- What are the real costs, benefits, and timetable going to be?
- Do you have an executive-level SCM champion that provides the necessary link to top management?
- Who will implement SCM and make it work?

Implementing a high-end SCM system for a large organization is a complex and lengthy project that requires a vast amount of resources (e.g., money, personnel, hardware, software, and communications network) and that could result in huge losses if it fails. In the case of SCM implementation, failure is not an option. It is a "do-it-right-the-first-time" kind of project, because most organizations will not recover from a failed implementation. Careful and meticulous planning and preparation is required for the successful implementation of the SCM system in the organization. Small organizations have options like implementing lightweight and open-source SCM systems or going in for source code repositories (discussed further in Chapter 20), which have a lower risk of failure and are easy to implement. Many of these options are free, and others are not very expensive.

This chapter describes the preimplementation tasks that need to be done to ensure a successful implementation. The preimplementation planning session is a multifunctional event that recognizes the SCM project on a formal basis. One of its primary goals is to develop the project plan. A project is a solution that is scheduled for success. The project planning session is a meeting that usually lasts one or more days and consists of all the critical stakeholders of the SCM system from both internal and external sources. The project planning session will define the main problems with proposed solutions. The planning session will determine what must be done, who will do it, how it will be done, when it will be done, its cost, and the services and materials needed to do it. The output of the project planning session is the project plan, which will serve as a guide to implement the software and help to build ownership in the system. The main tasks that should be performed during this session are listed as follows:

- Assembling the participants—stakeholders;
- Conducting a feasibility study review;
- Creating a project mission and vision statements;
- Determining the organizational structure;
- Determining the modules to be implemented;
- Creating the core team;
- Establishing the training needs;
- Establishing the data conversion or migration strategy;
- Establishing interfaces;
- Determining work estimates;
- Determining the cost of consultants;
- Calculating the implementation time;
- Identifying constraints;
- Establishing policies and guidelines.

Assembling the Participants

One of the first steps of the project planning session is to assemble the critical stakeholders of the project. This should include all people who have a direct influence over the project in terms of how the resources should be allocated. It should also include any potential members of the company who may participate or have an influence on the SCM core team. Participation in the project planning session is not an optional exercise. Because of the critical need for strong support in an SCM project, all members having a stake in the project need to participate. The full participation of senior management should not be overlooked. Any critical stakeholder who refuses participation in the project places the SCM implementation at risk, usually a high risk. The project planning session and the SCM project simply cannot take place successfully without the participation of the key stakeholders.

Feasibility Study Review

After the team is assembled a review of the feasibility study and needs analysis should be conducted. The feasibility study document contains the results of the investigation as to the feasibility of the SCM system in the organization. It contains the factors that will affect the SCM implementation—the ones that will assist and the ones that will create problems. The needs analysis will essentially be the justification for the project. The needs analysis document reports information such as why the SCM system is needed, which functional areas need maximum focus, and which areas need to be improved.

Project Mission and Vision Statements Creation

In this step, the mission and vision of the project is determined and documented. The vision should be a global statement that is continuous and ongoing. The mission statements will consist of the major milestones of the project. Mission statements should have specific expiration dates and be measurable in nature so that they do not become vision statements. They can be broken down by functional module. The expected completion date for each module should be documented as expressed by

the appropriate critical stakeholders. If they have not determined a critical completion date, then do the mission statements now and add the date later. The projected completion date will be calculated in a later step of the project planning session.

Determination of Organizational Structure

The organizational structure is determined to decide how the implementation is to proceed. Some organizations will be clearly separated by geographic facilities. Others may be clearly separated by different business divisions controlled by separate management functions. The purpose of this exercise is to show the major areas of the business that may install the software independently or at different times.

Determination of the Modules To Be Implemented

Using input from the needs analysis, vision and mission statements, and the organizational structure, the next step is to establish the modules to be implemented. Other independent software packages that may be interfaced may also be implemented with the rest of the SCM system. Include all possible sources that will integrate or affect the implementation of the SCM system. The modules of the SCM system may not correlate with the needs of the organization. It may be helpful or even necessary, to have an SCM system application consultant present to explain what functional need of the organization translates into the appropriate functional module of the SCM software. After the translation has been made, a complete list can be established.

Creating the Core Team

This session has several purposes. They include: (1) determining the structure of the SCM team, (2) assigning SCM core team members to modules and phases, (3) determining the team members' percentage of participation, (4) adjusting percentage equivalents, and (5) calculating full time equivalents. The type of SCM team structure to be used in the implementation should be decided. Other positions outside the core SCM team such as steering committees, management consultants, technical consultants, and application consultants can be applied and arranged to support the selected core team structure. Before one is chosen over the other, the advantages and disadvantages should be clearly understood by all critical stakeholders. After the structure of the team is established, by input from participants of the project planning session, each member must be assigned to a functional area and the percentage of time to be dedicated must be determined. The number of positions and the amount of time that SCM team core members will dedicate can vary considerably from the beginning of the project to the end. This can become quite evident when the project is broken up into phases and one person participates in more than one phase. Different phases may require different participation levels from different people.

Establishing the Training Needs

In this step, the educational and training needs are established. Here, education refers to providing information about SCM basics, best practices, needs and benefits,

different modules and how they function, and such general topics. The training imparts information specific for the company in that they will teach the users how to use the SCM system specifically for the business. Each training class will be custom-developed based on how the software is set up and used.

Determine how much education will be required, the type of education, and the number of participants. Educational classes will generally be predefined, standardized training packages. The focus of education should be the critical stakeholders, senior management, and SCM team core members. Some areas to consider for education include fundamental SCM concepts, business management techniques, SCM functional modules, and best practices.

Make a listing of all the participants who will be attending in each category. SCM educational modules are generally available from SCM vendors and service providers. They are standardized educational programs to help students learn the functionality of the software. Make a complete listing of SCM functional modules with each team member who will be attending. It may be helpful and even necessary, to have an SCM application consultant assist in the assignment of people to SCM functional educational training classes. Education for SCM functional modules should be further broken down by the phase in which the modules will take place.

When determining the total amount of educational needs, the end users should not be included in these figures. Some SCM team members will have the need for both education and training. For SCM team members who will receive both education and training, the educational portion should be included in the educational section, and the training portion should be included in the training section.

Establishing the Data Conversion or Migration Strategy

The goal in this step is to establish what needs to be converted or migrated and how it is going to be done. Almost every functional module will require information from the legacy system. The two primary methods of converting data are manual and electronic. When performing this step, it may be helpful to have an entity diagram and file structure specification of the SCM system. Application consultants can provide helpful insight into the critical files that require data for the successful use and implementation of the SCM system. In most cases, either electronic or manual methods may be used. However, before one is chosen over the other, the advantages and disadvantages of each approach should be clearly understood.

Some areas to consider for data conversion include open accounts receivable, open accounts payable, chart of accounts, accounts payable history, accounts receivable history, open sales orders, sales order history, open purchase orders, purchase order history, item master, item on hand balance, bills of material and routings, open engineering change orders, engineering change order history, open work orders, work order history, customer master, vendor master, fixed assets, project planning session, general ledger balances, employee master files, payroll history, and payroll balances.

Many other areas are also available for consideration. It is best to have experienced people from the legacy system working together with application consultants to fully understand the complete need.

Establishing Interfaces

The goal of this step is to identify any interfaces that require development. This section is for the required interface programs between systems for which none exist. If the SCM implementation strategy chosen is a big bang approach, it may then require few or no interfaces. Projects that are phased and have high-degree process characteristics need a large number of interfaces. Other software packages that need to connect and communicate with the SCM system should be included here. Interface programs can be developed using electronic methods or SCM team members or clerical staff can perform them using manual methods. Before one is chosen over the other, the advantages and disadvantages of each approach should be clearly understood. It may be helpful to use application consultants or technical consultants to explain critical interface points for the new SCM system. It is best to have company employees with a strong knowledge base of the legacy SCM system working together with knowledgeable SCM application consultants to fully understand the critical interface points.

Determining Work Estimates

The work estimates for all the activities from planning sessions to training and maintenance after installation of the SCM project should be determined in advance. A wide variety of events are represented that will consume the majority of the resources dedicated to the project. Performing work estimates in SCM systems is far from an exact science. However, if careful research was done prior to the purchase of the software, general estimates are possible with some degree of accuracy. Communication with outside references can provide estimates of how much work was performed and how long it took. Using these estimates from references combined with experienced honest application and technical consultants, reasonable numbers can be obtained. Estimates should be made for each activity for each phase or process section. The estimates for the work to be performed should be in days. The number of days is not based on how long someone thinks it will take from beginning to end, but rather on how long it will take based on a person working full time on that particular event. This is the equivalent of man hours or in this case, man days.

Cost of Consultants

It is better to establish the scope of the project or how much work needs to be done, in advance. With our previous section determining the available amount of resources, we can now move into our next section, which helps us determine the amount of time it will take. Most SCM projects benefit from some degree of consultation. This consultation can occur at management, functional, and technical levels. The amount of outside consultation that will be needed is dependent upon many different factors. Some factors that will increase the need for consultants include large complex installations, multilanguage characteristics, complex business process flows, lack of ownership, lack of participation by SCM team members, and

sudden changes in leadership. The amount of consultation assistance needed for an SCM project can vary tremendously. When and where a consultant may be needed is very difficult to predict in the early phases of an SCM project. For purposes of increasing ownership and participation, it is better to limit the role of consultants and use them for critical areas.

Calculation of Implementation Time

Three characteristics working together—scope, resources, and time—will affect every SCM project. The scope of the project has already been defined in the previous steps. The available resources have been calculated through percentage equivalents into full time equivalents. The remaining step is to calculate the completion date. If a company increases the scope of its SCM implementation, then it will need to increase the time or available resources, or both. Similarly, when a company decreases the available resources used to work on the project, then the scope will need to be decreased, or the available time will need to be increased, or both. By changing any one of the three characteristics the other two are affected.

Once the scope, time, and resources have been calculated, a graphical timeline should be developed using a variety of project management software tools. The timeline will represent a high level showing the major phases or processes, activities, sequence of activities, events that will operate in parallel, start dates, and completion dates. A graphical timeline is an important part of the project planning session. The graphical representation of the elements of the project helps make it a valuable tool for discussion.

Identifying Constraints

All constraints of the project should be identified and documented. A constraint is any business activity or external factor that will inhibit or affect the outcome of the SCM implementation in some negative way. The source of information should come from all the critical stakeholders of the SCM implementation. One of the key factors on which to focus is identifying any activities that will potentially compete for the time of SCM team members. Some examples of business activities or problems include lack of people, data integrity problems, people unwilling to change, other implementations, change of personnel, and new products or services.

Establishing Policies and Guidelines

Project policies and guidelines form consistent methods for dealing with situations and events that occur in a project. Some areas in which to consider policies include problem resolution, software modifications, configuration settings, business process flows, meetings, budget changes, resource acquisition, general communications, conversion strategies, contingency plans, project status meetings, training guides, and user guides.

In-House Implementation—Pros and Cons

A question that many people ask is why can't the company carry out the SCM implementation by itself? To successfully set up and implement a perfectly functioning SCM package is not an easy task. One cannot go in for a trial and error method of implementation strategy due to the huge investment involved. The consequences of a failed SCM implementation can be quite catastrophic. It might put the organization out of business. Also, the SCM implementation process cannot go on for a long time. It has to be completed within a reasonable time period.

To successfully carry out the implementation within a reasonable time frame, the in-house people who are designated to do the job should possess certain knowledge and skills. To start with, the company should have many people who are experts with the SCM package and with technical issues. Implementing the SCM software means assigning the optimum values to the various parameters and variable elements of the system. Experience has proven that a good professional needs at least one year to become reasonably good in operating an SCM system, and this should involve hands-on practical experience. As you cannot become an expert by reading product brochures and on-line help files, you have to have practical implementation experience.

Many software vendors have their own team of consultants whose responsibility it is to implement software packages following a standard approach or methodology. Definitely, these people know the product and can be of great value during implementation. However, developing a good software package and successfully implementing it are two entirely different propositions. So, a good package vendor need not be good at implementing its product. Also, each group of people in an implementation project (e.g., vendors, consultants, in-house team, and users) has a definite role to play in the implementation. So if the same party is performing multiple roles, it can create problems when a conflict arises. For example, if the vendor is doing the implementation, the vendor's consultants may not be as open to the ideas of the in-house team as third-party consultants, because the vendor's consultants will have a mindset that will prevent them from seeing the other side's perspective.

Besides having a very good knowledge of the product, the people who are to implement the SCM system should possess the following skills:

- *Knowledge of how to organize and run a project of this magnitude.* This calls for good organization skills, project management skills, team management skills, and knowledge of scientific methods of software project management.
- *Experience in handling problems and issues that arise during the implementation.* No implementation will be a smooth process; there will be problems such as cost overruns and time overruns. Knowing what to do in these situations is vital for the success of the project.
- *Good people skills.* Any SCM implementation will face resistance from the employees. The resistance could be due to many factors including ignorance of the product, fear of unemployment, fear of training, and fear of technology.

So, it is important that the people in the implementation team be good diplomats adept at diffusing crisis situations.

- *Good leadership skills*. SCM implementation will involve dealing with a lot of people, and good leadership and communication skills are very effective.
- *Excellent training skills*. Every SCM project involves considerable amount of training at various levels and in various details. There will be familiarization programs for all the employees; there will be executive programs for top management; there will be functional training for the implementation team members; and there will be end-user training.

In today's business environment where the trend is to reduce manpower and focus more on the company's core competencies, it becomes even more difficult to take the total responsibility of the SCM implementation and get it done using in-house resources. If the company is planning to do the SCM implementation in-house, it might have to hire experts and have them on the company's rolls. This is an expensive proposition because once the implementation is complete—the post-implementation phase—you will not need that many experts to keep the system running. You will need, only a handful of people—perhaps just a few of them in each functional area—to effectively handle this postimplementation scenario. So, if the company is planning to do the SCM implementation all by itself, then it will be adding a lot of resources and spending a lot of money on training, most of which will not needed after the package implementation.

It is therefore, always a better idea to leave the implementation—or most of it—to the people who are specializing in that and concentrate the company's efforts on preparing its personnel to administer the package after it is implemented. Once these employees have been trained—during the course of the implementation—they can help the company in its implementation efforts in other units of the company or provide training to other employees in using the system, among other jobs. So by getting the employees trained during the implementation, the company can save a lot of money it may otherwise have spent in hiring trainers.

In summary, it is better for companies to concentrate on their business and leave the job of SCM implementation to people who are in that business. However, to get maximum benefit out of a packaged solution, company personnel should participate fully during the implementation of the package. The company should plan the participation so that its people play an appropriate role in the implementation project and will have enough experts in-house, once the implementation is over. The company should bring in the "know how" and experience that will guarantee the best possible use of the acquired package.

SCM Implementation Plan

Before implementing an SCM system in an organization or project, it is crucial that the implementation process be planned. Implementation planning includes developing the implementation strategy and implementation schedule and organizing the implementation team.

The implementation plan documents the who, what, why, where, when, and how of the project. It is the outcome of discussions with affected people and involves negotiations over resources, timescales, and costs and their agreement. It should be realistic; otherwise, if timescales are too short, potential disruption will be built into the plan, and if timescales are too long, the momentum can be lost. The plan provides a guide to the project and is used to monitor progress. It enables people to carry out a set of interconnected tasks in a coordinated manner. Setbacks are highlighted and remedial action established. If necessary, dates are rescheduled. Importantly, the plan is communicated to all who need to know about the project, making them aware of progress and changes.

The implementation plan should address all concerns such as existing procedures, the effect of the SCM implementation on the procedures, how it will affect the employees, the work environment of the organization, the SCM awareness of the employees, and the creation of SCM user manuals.

The most basic plan will identify all the activities, those doing them, and the time frame. A project plan can be handwritten or produced using some computer application. Spreadsheets offer simplicity and are readily available. Furthermore, they can easily be distributed to others since a spreadsheet tends to be a standard tool on many PCs. Alternately, specialized packages are available such as Microsoft Project. These capture a lot of detail about the project, enable different views of the project, such as timescale or critical path, and facilitate the reporting of many different issues (e.g., costs, resource usage, and overdue activities). The key concern is the amount of complexity to be organized, manipulated, and updated—a function of the amount of detail required. So it is necessary to ask what detail is going to be useful. Another consideration is the distribution of the plan. With a specialized package, not all intended recipients may have the required software, raising the dilemma of what to distribute and how.

A project plan outlines the major tasks, the estimated duration (usually specified in months), the resources required, and the people who will be doing the tasks. The high-level plan gives an overview of the project and can be used by top management for monitoring the project. However, this high-level plan cannot be used by the person who is responsible for day-to-day activities of the project—the project manager. The project manager will develop a detailed project plan, where the high-level plan is broken down into a lot more detail with the time windows being weeks or days rather than months. Additional columns may be used to identify start dates, end dates, amount of work (hours), estimation of percentage completion, lateness, costs, and any other issues deemed relevant. Some tasks will be dependent upon the completion of others (interdependencies), while other tasks can run in parallel (concurrent engineering).

Timescales should be realistic. If they are overly optimistic, then the project will soon fall behind and is unlikely to catch up. This will have the effect of demoralizing the team. Likewise if the timescales are too long, momentum may never build up, and delays may result from inertia. Having produced a plan, it is then important to maintain that plan against progress, revising it as necessary. When problems arise, so do the potential for delays. It can be argued that it is better to push out dates rather than not get it right, since a sloppy solution may reemerge at a later date with a magnified impact.

Risk Assessment

Even the most detailed of project plans can go astray for events that could have been anticipated and prevented. Accordingly, it is prudent to carry out a risk assessment. The aim is to anticipate possible problems, to assess their likelihood of occurrence and their intensity of impact, and, finally, to establish how they can be prevented or best handled if prevention is not possible (e.g., causal analysis).

There are various approaches to risk assessment. For many, the prerequisite for understanding what is involved in a risk assessment is denied by the virtue of never having been here before. To keep it simple, one should adopt a basic approach, starting by recognizing that the project has the potential to fail. The first question that should spring to mind is: why?

The first task is to understand what is involved. Risk assessment should be done at the first possible opportunity. It should be done by an individual or team of experienced professionals. An appreciation of what is involved will enable assessors to determine potential risks. An insight into potential issues can be gained by reviewing the main problems experienced by others. Many tend to be people-related. Technology and methodological issues tend to be of lesser prominence. The result may be an unwieldy, long list. In this case it may be desirable to establish which risks are the most important and to focus on those. Each risk is assessed for how severely it can impact the project and the business and the likelihood that it will occur. This is a subjective process, and the views of others can aid this. If a risk has a high severity and a high likelihood of occurrence, it requires immediate attention. Likewise, risks rated with low severity and a high likelihood of occurrence will require attention if they are not to be disruptive. Cases where there is a low likelihood of occurrence can be put to one side. Note that they are not discarded. This process prioritizes those issues that need attention. The result should be a reduction in the likelihood that things will go wrong. However, while it is proposed that this assessment is carried at the outset of the project, it should be regularly revisited. Process developments and changes in project conditions may raise the profile of risks that were previously viewed as insignificant.

Implementation Strategy

One of the key decisions is the scope of the implementation. One option is to undergo a complete switch-over from any old systems to the new system, the "big-bang" approach. Another option is to introduce the software in stages, with core functionality first and additional functionality rolled out in successive stages. In both situations, there is the option of whether to have both the old and new systems run in parallel. Consideration should be given to a number of factors:

- Speed or urgency of implementation;
- Availability of people for carrying out the implementation tasks;
- Availability of time for training all users;
- Cost;
- Confidence in the new system;

- Disruption to operations;
- Total timescale.

Whichever option is decided, those involved in the implementation must remember that they will spend a lot of time on development work, time that would otherwise be spent on normal duties. Thus, the question arises about how these normal duties are to be fulfilled. One option is to recruit temporary staff to carry out normal tasks. However, this assumes that the required skills can be acquired. An additional option may be available for those programs or organizations that are running multiple projects: a pilot project. The organization would select one project that could be representative of the others, ideally a project that is not too large, and implement SCM for that project—running the SCM pilot as a project unto itself.

The implementation plan should identify the implementation strategy, that is, whether the incremental approach or the big-bang approach will be used and whether SCM will be implemented in all projects simultaneously or instead introduced in a pilot project.

When the incremental approach is used, full SCM functionality is not implemented in one step. The different functions are introduced one by one. For example, the organization might choose to implement the change management system first and then the other functions at a later date. The advantage of the incremental approach is that the company can get feedback on the implementation and how it is received and possibly fine-tune the implementation strategy based on the feedback received. Another advantage is that it can spread the investment over a period of time.

The advantage of the big-bang approach is that the company can start reaping the full benefits of SCM soon after implementation. The big-bang approach is effective if the organization is mature and conducive to SCM, and personnel are ready for the SCM. Here, there is no room for error—it is a "do it right the first time" proposition. Implementing SCM in a pilot project is a good idea, because it will give the implementation team a feel for the issues in an actual implementation, the peculiarities of the organization, and its work environment. A successful pilot project can be used as an effective tool for convincing staff and alleviating any doubts and eliminating fears about SCM. However, the pilot project must be selected carefully—it should be a project where the team members are willing to face the challenge of doing something new, have an open mind, and can adapt to new systems. There is nothing like a successful pilot project implementation to convince others why they should also implement SCM in their projects.

Budget

With the costs identified (during the planning stage), a budget can be established. It should also be anticipated that problems and unforeseen issues are likely to result in additional expenditure. Whether an allowance is made for this is a policy decision by the finance management. Actual expenditure is monitored against budget for the duration of the project. Variances to be aware of include high consultancy costs, particularly in the early stages of the project, and low training costs. Details

of consultancy costs should be identified during the selection process and monitored to avoid overspending. Unless the project specification has changed between then and the implementation, this estimate should roughly reflect what is incurred. If there is a significant variance, it is necessary to ask why this has occurred. The other major cost to monitor is training. When there is an indication that budgets are going to be overspent, it tends to be the training budget that suffers. However, training is one area is often reported as being inadequate.

Cost

It is unlikely that a financial director would support the idea of unlimited funding for an SCM implementation project. Instead, from a control stance, a budget needs to be established. This will be based upon an estimate of the likely costs. In identifying where the costs are likely to arise, consideration should be given to the following:

- Hardware;
- Operating system;
- Database license fee;
- Core software license fee;
- Additional module license fee;
- Additional seat license fee;
- Third-party software license fee;
- Integration of third-party software;
- Software customization;
- Project management;
- Consultancy;
- Training;
- Living and travel expenses (also travel time);
- Software maintenance or warranty renewal;
- Upgrades.

Much of this cost information will be provided by the vendor. While it is likely that the vendor will provide a specific figure for each item, this may only be an estimate. Whilst the costs of the software can be precisely stated, where there is uncertainty about what is involved, the cost will only be an estimate. This is likely to be the case for such items as consultancy, which history suggests is an area for potential overspent. In this case, an upper and lower value, and the expected value should be sought to give a fairer reflection of the potential cost exposure.

While some of these costs will be one-off (e.g., hardware, training, and consultancy), others will be ongoing (e.g., maintenance). To get a better picture of the cost exposure, a long-term perspective should be taken. A meaningful time horizon is five years. By the time that five years has passed, it is quite possible that the application will have been reviewed and a new budget established for additional work, such as an upgrade or additional functionality.

Not to be overlooked are the indirect costs, which are mainly internal costs. These can include the following:

- Time and consequent cost of employees involved in the project;
- Cost of temporary personnel to replace those involved in the project;
- Cost incurred due to other activities not being carried out costs related to off-site travel and sustenance (e.g., off-site training);
- Costs related to the internal resources, such as the implementation team or work team, who administer and maintain the system and provide internal technical support.

The annual maintenance fee may be surprising in how significant a proportion of the total budget it is. Is it worth asking what is provided for this fee? Obviously, each situation will be different depending on a host of variables, including the complexity of the processes affected, the number of users, the amount of customization, and dependency upon consultants. It should be possible to pin down most of the costs, especially the main costs, to a lower and upper value and also a most likely cost. However, the unexpected can disturb this picture. Some costs will not be readily apparent and can be overlooked. Alternatively, they may be underestimated. It is not uncommon for the yearly cost to be in the neighborhood of 10% of the initial cost outlay.

The danger arises when a budget is set, but costs during the project continue to escalate. This is particularly true with regard to consultancy and training costs. Overspending on consultancy is often compensated for by a cutback in training. This is not helped by the fact that training costs tend to be underestimated in the first place. The dilemma faced is that having started the project, so much has been invested in it that it must finish—but at what cost? This raises the need for cost control throughout the project. Project planning software may have the facility for attaching costs to resources used in activities planned. This can provide a means for determining indirect costs. Upon completion of the tasks, the plan is updated to reflect the actual resources used and the time taken. This gives an indication of the actual cost incurred. It is particularly useful for gauging the costs of internal personnel.

From a practical point of view, the one area where costs are most likely to get out of control is the use of consultants. On-site attendance of vendor personnel can boost costs if uncontrolled, or the unexpected occurs. Thus, it may be desirable to contrast the estimated costs of personnel on a "time and materials" basis with a fixed-cost implementation. While the latter may be more expensive at quotation stage, the reality may be the opposite. Typically, SCM tool consultants (whether independent or the vendor's representative) will cost in access of $1,000 per day.

Cost-Benefit Analysis

Once the need has been defined and the costs identified, it is useful to determine what the benefits are and whether the benefits justify the cost. This justification can strengthen the argument about the need. However, the dilemma arises as to how to carry out this justification. The SCM implementation is by its very nature

a complex activity. It involves many people who need to act over a long period of time in a coordinated manner to produce a coordinated way of working using a technology that may not function precisely as desired.

A similar case can be made for the determination of the benefits. While the tangible benefits of reduced stock holding may be readily quantified, the increased revenue resulting from more efficient operations will prove difficult. Furthermore, the quantification of any intangible benefits will prove difficult. However, from a practical viewpoint this exercise is not required to be, and cannot be, an exact science. The aim is to get a useful picture of the situation in terms of what to expect. Through the application of models, such as "six sigma," the company should see a considerable drop in software defects as a result of implementing SCM activities.

The identified costs are assessed within the context of what is understood about the future and the benefits likely to be gained. With a time horizon likely to be five years, much can happen in that time. Thus, the approach should be simple with assumptions clearly defined. This analysis can also be used to establish reference points or benchmarks. These can be used to assess progress and whether the anticipated benefits are achieved. Furthermore, it is prudent to be cautious about potential gains. History suggests that costs will overrun, and benefits will fail to materialize. This is aside from the skill required of the project manager in controlling both the costs and the realization of the benefits.

Finally, consideration must be given to the context within which the justification is being used. Who is producing the justification? How will others use these benchmarks—now and in the future? If the situation changes in the future, then the relevance of a specific benchmark may change. All too often, planners cast forecast numbers into stone, yet ignore when conditions change, making these numbers nonsense.

Performance Measurement

The notion of performance is associated with the concepts of control and targets. We have already seen three performance related measures—costs, time, and benefits. However, an SCM implementation has a notorious reputation for being overspent and late and failing to realize benefits. Furthermore, with the timescale long, it is not practical to wait until the operation phase to find out if everything is functioning as intended.

The project plan identifies what tasks need to be done. The aim is then to carry out the tasks. It is desirable to have some indication that tasks are taking place as required and that they have accomplished what was intended of them. The assumption is that when a task is done, it achieves an objective(s). Thus, the training of a user should result in a user being able to perform a set of tasks better or gain a better understanding of SCM concepts.

For each step or series of steps of the implementation, objectives can be defined that, if achieved, represent progress. By achieving these deliverables, there is less likelihood of problems arising at a later date as a result of an earlier event. Conversely, failure to achieve these deliverables and the subsequent progression to the next stage will increase the likelihood of potentially significant problems arising

at a later stage. Furthermore, progress can be monitored in a methodical manner. Each task or set of tasks is evaluated as to its successful completion (e.g., whether training has been effective, whether documents are complete, or whether processes are fit for the purpose).

Together, the four measurables (cost, time, benefits, and deliverables) present different dimensions for measuring the performance of an implementation. Often, it is only the cost and time dimensions that are monitored. Understandably, these have a visible effect on the finances and operations of the business. Rarely are the benefits assessed. It can be argued that monitoring benefits is carried out after the event, so what is the value of monitoring beforehand? However, by assessing whether benefits have been achieved or not (and the reasons), an opportunity is created to learn from what has happened. These lessons can then be applied to further phases in the implementation.

The use of deliverables provides the opportunity to assess the effectiveness of what is being done. However, conflict may arise. Ensuring that a task is properly completed may involve an unanticipated increase in the amount of work and lead to it being late. While the deliverable has been met, it is at the expense of two other measurables—time and cost. So when it comes to determining whether the project has been a success, consider which measurable is being used. This dilemma is magnified when targets are misused as a tool to blame someone or to score points over others. It is important to remember that while measurables provide a means to assess progress and attainment, they in themselves do not determine success. They merely provide reference points for further action. They are not a substitute for managing people in such a way that they give their best and more.

SCM Implementation Team

The SCM implementation project is a joint effort of many groups of people—the in-house implementation team, the package vendor's team, outside consultants, end users, and company management. This section discusses how to organize the internal resources of the company for an SCM implementation.

The most frequently asked question is, "Who within the company should participate in the project?" The natural response is everyone who is involved in the SCM process—developers, QA team, project leaders, support personnel, marketing team, and top management. Because the SCM system is an integrated package, almost everyone in the company must participate in one way or another. The functionality of today's sophisticated SCM systems will extend to practically every sector of the company.

The usual organization of the implementation team takes the following format. The person who manages the implementation is the project manager. The project manager reports to an executive committee, which reviews progress and resolves any territorial, resource, or policy disputes. The CEO or MD leads the executive committee and sponsors the project.

Working for the project manager are the members of the project management team, who carry out the various tasks for implementing the system. They are the

people who set up the infrastructure, produce documentation, and train employees. The tool vendor will appoint one or more of its consultants to provide support to the project manager, manage the client account, and coordinate other vendor resources. Vendor consultants advise about best working practices and software functionality and assist with technical issues. Training is provided in the first instance by the vendor to the project management team through either the consultants or specialist trainers. Once the project team has developed and proven the new way of doing things, it produces the procedural documentation and trains other employees.

However, just because everyone in the company should be involved in the project, we do not mean that everybody should stop their jobs and join in the SCM implementation effort, thus virtually stopping the company's day-to-day functioning. Still, the company will need an owner or sponsor for the project—somebody who is going to lead the implementation. This should be a senior person who has knowledge of the SCM system and the organization and the necessary authority to make decisions and implement them. Then the company will need at least one professional from each major department to carry out a specific project function. Roles will vary in complexity and time involved, but all are equally important for the success of the implementation.

It is very important for all business functions having some relation to the SCM system to be represented adequately in the project team—from the top level to the lowest level of operation. This is important because these are the key people who will make the SCM system acceptable to everyone in their own departments. So if the implementation team is comprised of people from all departments, from all levels, they can convey what they have learned about the SCM system and help overcome the initial resistance the system is bound to face.

The implementation of the SCM system demands that the managers and operational staff understand how the new information environment is going to work. They should be given a clear picture about what is to be changed in the current setup and what additional facilities the new technology will give the end user. This is important because the new technology and the additional responsibilities that arise when the new system is in place can overwhelm many people. So it is important to give everybody involved an idea of what to expect before the project starts.

For the right level of participation to occur, a group of representatives selected from various departments of the company will need to involve themselves in several levels. Some members of the project team will work on the project full-time—in tandem with the in-house experts, external consultants, and the vendor's team. Others will help coordinate the tasks of the different sections and make available all the resources required to the implementation team. Management representatives will monitor the progress of the project and make decisions and corrective actions to keep the project on-schedule and within budget. Other members will participate part-time during workshops and training.

One important thing to keep in mind is that the effective participation of in-house personnel is not possible without the full commitment of top management. It is the responsibility of top management to see to it that the people who are designated to the implementation team on a full-time basis are not interrupted in any other way or with any other work.

Composition of the Implementation Team

Who should be assigned to the implementation team? This is a very important question, because the success of the implementation and the continued functioning of the system depend on these people and their ability to grasp the new tasks and technologies.

Once the SCM system is implemented, the current processes and procedures will be replaced by new ones. The job descriptions and responsibilities will undergo some changes. Information integration will happen, and many processes will be automated. An action taken by an employee can trigger a lot of procedures and affect a lot of other functions—almost instantaneously. The technology will bring with it a series of new concepts and resources that must be mastered and correctly used to get the best out of the SCM system. The format, the speed, and the content of the management information systems will change. The decision-making process will change because the decision makers will be able to get accurate information, in the form they want, when they want it. The SCM packages will institute new development models and practices.

So the company should appoint its best and most efficient employees to the implementation team. The company should invest in these people and create opportunities for them to excel within the company so that they can grow with it. At the same time, however, these are the people who are actually running the business, who do not have time for anything else, and who everybody will turn to in a crisis. Nevertheless, it is these people who should be assigned to the implementation team.

SCM implementation is a very complex and sophisticated project, involving technological as well as cultural changes. It is not the place for people without initiative, dedication, and enthusiasm. It is not the place for people who do not have any team skills or who have communication problems. It is not the place for people whom the boss does not want. In fact, assigning some people just because they are the only ones available is one of the worst mistakes that management can make. Such an action could jeopardize the entire project. The SCM implementation project needs people who can grasp new ideas quickly, who have an open mind to new technologies and concepts, and who love challenges. These people should have a never-say-die attitude and should be capable of working as a team.

These people will have a lot of demands placed upon them. They will need to develop a detailed understanding of how the software works, pick up new skills, such as process mapping; carry out totally unfamiliar tasks, such as prototyping and training; and deal with problems that they normally would never encounter, including establishing which process path is the more acceptable. These men and women will be pioneers as they take their organizations through untested environments and uncharted waters, so their ability to think quickly, improvise effortlessly, innovate fast, and act without hesitation is critical to the success of the project.

So when faced with the decision of assigning members to the implementation team, management should be willing to send their best staff members. Invariably, these people are those whose work cannot be interrupted and responsibilities cannot be delegated. Nevertheless, the company has to find a way. If the company decides—early enough—who they are going to send, and if those people are informed, they might be able to train replacements to do their work until they return. *Sending the best people is worth the effort.*

Those selected—the pioneers—will have greater demands made upon their time. They are likely to work long hours for many months, including weekends, to cover their normal duties and their project tasks. During this time, it may be necessary to restrict when they take their holidays to avoid their coinciding with a critical stage in the implementation. The continual intrusion into free time may affect family life and needs to be accommodated. For this reason, it would not be unreasonable to review remuneration or holiday entitlement for team members. It should be acknowledged that people have their own learning styles and different rates of working, particularly when dealing with the unfamiliar. As time passes, team members need to be watched for loss of interest, resentment, or burnout and handled with care. If team members leave the company, bringing replacements up to speed will cause delays.

Organization of the Implementation Team

Figure 17.1 shows the organizational chart for a typical implementation team in medium to large companies. For a small company, there is not much difference in the organization except that the team size will be smaller, and the executive committee and the project management committee might be merged. At the top of the chart is the executive committee headed by somebody from top management. Then there is the project management team, followed by the technical and administrative support personnel. Then we have the work team. Let's look at the functions of each team in some detail.

Chief Executive Officer (CEO)

The most senior role is that of the CEO. This person has two key responsibilities. The first is to promote the vision of what can be achieved using the implementation as the opportunistic catalyst for change. How radical this vision is depends upon the individual and also senior colleagues, for these colleagues should share and make their own contribution towards this vision. The vision of what needs to be achieved must reflect what the organization's capabilities. Irrespective of whether

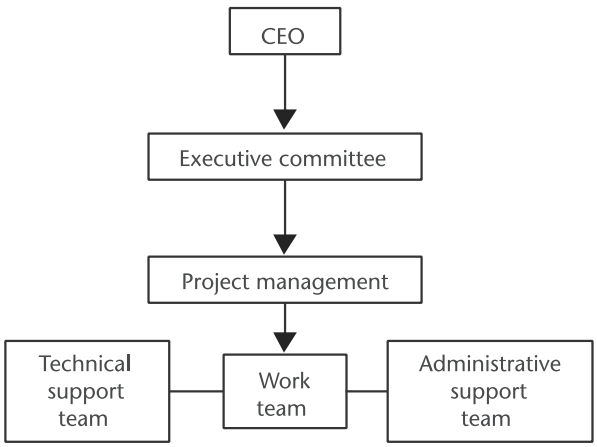


Figure 17.1 Organization of the SCM implementation project.

the organization has the capability to move mountains or only molehills, its focus is upon progress. However, the greater the ambition, the more commitment is required. This is particularly true in the case of senior and middle management, who, in a changing environment, are traditionally the most resistant. It is the CEO's role to assess this capability and lead accordingly.

The other responsibility is sponsorship of the project. This means that the CEO needs to give support to the project in such a way to promote its importance, support the position of the project manager, bring into line dissenting middle and senior managers, act as arbitrator, and carry out any other activities that ensure that the project does not flag, including replacing the project manager if necessary. To avoid cynicism, the CEO needs to ensure that the gap is bridged between speech-making and action.

Executive Committee

The executive committee is a group of senior management personnel that represents the interests of the company management and that is headed by the person who is in charge of the SCM project implementation (sponsor), a person in whom the company places its highest confidence—someone who is considered a leader, who has the necessary authority, and who carries enough clout within the company. Usually, the CEO or a very senior manager will be the sponsor for the SCM implementation. In the case of large corporations, the CEO or top management may not be members of the executive committee as they will be busy with strategic planning and decision-making. In such companies, the responsibility of SCM sponsorship will be entrusted to somebody in the top or middle management, usually to somebody who reports to the chief technology officer (CTO) or chief information officer (CIO).

Irrespective of the title of the sponsor and the executive committee members, their primary function is to make sure that everybody knows that SCM has strong management backing and support. Also, these people should have enough authority (or direct access to people who have authority) to make and implement decisions that are required for the smooth progress of the implementation.

The committee should also include the external consultant's representative and a consultant from the package vendor.

The executive committee defines the objectives, monitors progress, and quickly resolves issues that are brought to its attention. It ensures that the conditions are right for the implementation. This means that the committee eliminates any possible conflicts of interest the project team members may experience. It also means that the project manager is able to escalate concerns to the executive committee, that the concerns will be acted upon, and that the extra time and effort of those involved will be recognized. The executive committee also ensures that corners are not cut for the sake of convenience and is committed to removing any barriers that may hinder progress and the realization of benefits. The committee should be aware of, and adjust for, possible discord between the discipline of project management and its own management style, particularly if project management is not a familiar practice within the organization.

The executive committee is responsible for monitoring and evaluating the project and its progress. The committee is the body that approves budgets and initiates

corrective actions when things are not going according to plan. So the committee should establish a reporting and monitoring mechanism by which it will be kept abreast of progress. The monitoring mechanism should have facilities to alert the committee about impending disasters and delays well in advance so that corrective and preventive measures can be taken. The committee should meet at least once a week and in the event of emergencies. The project manager reports to the executive committee at the weekly business review meetings to enable required discussions to take place.

External Consultants

The role of external consultants in the implementation is to advise the project manager and the implementation team in all areas where there are no in-house experts. The cost of using the external consultants is a huge drain on the implementation budget and should be kept to a minimum. If a company is already using an SCM system, has previous experience in implementing SCM systems, or has employees with the necessary expertise, then it can limit the number of external consultants to the minimum. However, in the case of companies who are doing an SCM implementation for the first time and whose employees do not have much experience, it is absolutely a must to get professional help; otherwise, it may face such consequences as failed implementations and poorly designed and implemented SCM system. In conclusion, the principle should be use external experts when the company does not have in-house expertise; otherwise, use the in-house experts.

Project Manager

The pivotal role in an SCM implementation project is that of the project manager. The project manager is the catalyst—he or she makes things happen. A project manager has a can-do attitude and is a communicator, diplomat, and facilitator who is knowledgeable about the business, credible within the company, impervious to criticism, and resilient. The project manager is also an administrator, keeping records about project progress, maintaining the project plan, handling correspondence, and checking invoices.

Project managers can become ineffective if there is no demonstrable support from the top. Also, project managers may need things from people over whom they have no authority. If they are well-liked, they may sway people to cooperate. However, problems will arise, as in the case where members of the project team split their time between their normal duties and those of the project. Operational demands will prevail when it comes to deciding which takes priority. Likewise, decisions about tasks may lead to the desire to shift work from one department to another. While it may make sense to make this change, the departmental head may object. The department may already be overstretched, and head count may be an issue. The sponsor must deal with these and other predicaments. Thus the roles of project manager and sponsor are complementary and essential for success. There are cases of projects that have progressed well and then stumbled following the sponsor's departure. Tasks that should take days take months, because the project manager does not have the status to complete the tasks that the sponsor has.

Project Management Team

The project management team is headed by the project manager and is comprised of the technical leader (leader of the external consultant team), the vendor's project manager, and the implementation project manager. The project manager will report the project's progress, problems, and other issues to the executive committee.

The project management team is responsible for conducting the scheduled work, administering the project, and communicating with the in-house team and the consultants. The team members should monitor the implementation team's progress, assess the amount and quality of the contribution of the team members—both in-house and consultants, and resolve issues that exist. Since the project management team has the project manager, the consultant team's head, and the vendor team's head, most problems can be resolved at this level. If any problems cannot be resolved at this level, then the project manager will escalate it to the executive committee for resolution.

The project management team should also ensure that company personnel and consultants are working together as a team and that there is full cooperation between the two groups. They are also responsible for ensuring that the consultants are transferring their knowledge to the in-house team and that all the documentation is done properly. The project management team should make sure that even after the external consultants and vendor representatives leave, the system will run smoothly.

Work Team

The work team is comprised of people who will actually perform the tasks set forth in the project plan. These tasks range from migrating the project information to the new system, to user training, to monitoring the start-up of the new system. The people on this team should be the best in the company and should dedicate their full time and attention to the SCM implementation project.

Team members need a knowledge of the company's work culture and environment, awareness of company policies and regulations, good analysis skills, team spirit, a cooperative attitude, good communication skills, patience, persistence, self-confidence, and, above all, sound common sense.

The work team normally includes hired consultants and the in-house team. These consultants should have a good understanding of the software that is being implemented. That is why they are hired in the first place. These consultants most certainly must have participated in the implementation of similar projects before.

The company's in-house team is the people with the knowledge of how the company works. They are the people who are going to use and run the system in the future. They and the consultants together decide on how the system should work. The in-house team members will be the first people to receive training on how to operate the software. They must know how the system works to evaluate the impact of the software on the company's current business processes. They will also discuss with the consultants and the package vendor the level of customization the product will require to function properly in the company. The work team will test the system once it is installed. The work team will also participate in the

training of the end users of the system. The in-house team will contain people from the company's various functions or departments. The work team will also have hardware engineers who will manage the technical requirements of hardware, network, and software.

The work team manager should hold regular meetings that bring the team together so that they are all aware of what is happening. This is an opportunity for progress to be reviewed, for issues to be highlighted and for problems to be shared.

Consideration should be given to the team's work environment. The ideal setup is a dedicated room where people can work undisturbed. This room should have plenty of wall space so that charts can be hung on the walls. It should be equipped with networked PCs, tables, shelving for documentation, and flip charts. Laptop links should be available so that consultants can link their equipment into the network. This room will serve as the nerve center for the project. As well as a work environment, it will hold project meetings and serve as a training venue. This room need not be abandoned after the implementation is over, since it can serve as an operations room for continuous improvement teams and as a training venue for new recruits, those wishing to upgrade their skills, or those wishing to test out refinements to processes.

Technical Support Team

The function of the technical support team is to create an environment that is suitable for the implementation of the software. The team size of the technical support team is directly proportional to the size of the work team; usually the team size is three or four. This team works very closely with the work team and takes care of issues like data migration, data backup and recovery, hardware infrastructure, and performance tuning of the databases.

In short, the technical support staff is responsible for ensuring that the machines will be up and running, the network is functional, and the hardware infrastructure is in good shape for the work team to implement the software package. These are the people who will be doing these activities once the implementation is over and the system is live. So they should interact with the consultants and the package vendor to assess any special arrangements or hardware or maintenance and backup and recovery procedures that may be required for the system.

Administrative Support Team

The job of the administrative support team is to make the life of all others on the implementation team easier, so that they can concentrate on their tasks and be more productive and efficient. Here also the team size will be three or four. The support teams' responsibilities include making available the workspace, tables, conference rooms, telephones, stationery, filing cabinets, and any other resource required by the project team (including, of course, refreshments!). Other duties include arranging the meetings and conferences, making photocopies of documents and circulating them to the right people, and any other administrative tasks that make the life of the work team easier.

We have reviewed the organization of the SCM implementation project team. Each implementation project is different and will have its own characteristics. As a result, there will be some changes in the exact constitution of the teams, but the basic components should be there for the proper functioning of the implementation project.

How the Implementation Team Works

So far we have seen how the project implementation team is organized. At the top is the executive committee, and one of the main responsibilities of the executive committee is monitoring and evaluating the progress of the project. The executive committee should develop a management and reporting mechanism so that members know what is happening to the project on a regular basis. This section examines how this is done.

How can the company establish an information base to determine whether the work is progressing according to plan? How will the company decide that the present course is the correct one and will lead to the successful completion of the project?

One of the main roles of the members of the executive committee is to check and verify that the work that is being done is satisfactory and that the momentum, morale, and enthusiasm of the work team who are per forming the tasks are maintained. During executive committee meetings, the members should receive reports and other information from the project managers as to how the work is progressing and whether everything is going according to schedule. The executive committee should receive data that induces it to maintain confidence in the implementation process.

Before the implementation starts, the external consultants and company representatives prepare a work plan. This plan details each and every activity that needs to be carried out and when they should be carried out. The consultants should lead the process of work plan preparation, because they have experience implementing the same package in similar conditions. The in-house team should point out the issues that are specific to the company and help the consultants create a realistic work plan.

The work plan or the project plan forms the basis for project tracking and monitoring. The project plan contains numerous activities, the person-hours required to complete them, and the resources needed to perform the tasks. The project plan is often built using a project management package (such as MS Project) that permits one to focus on planned activities from various perspectives—the chronological sequence or timetable, specific activities and who is responsible for them, the prerequisites for carrying out a specific task, or a PERT chart of the activities. Preparing the project plan using such a tool helps improve the quality of the plan and makes it easier to make changes and adjustments.

Once the project is under way the plan can be updated on a regular basis and the "planned versus actual" reports can be produced in varying detail and in varying formats including graphical. The project management software can generate comparisons between the actual and planned completion dates, expenses, and other similar criteria. These software tools allow responsibilities to be assigned to different persons—so it is easy to find out who is lagging behind.

Keep in mind, however, that, irrespective of whether the plan is created manually or by using a software package, all the parties involved—the executive committee, the vendor, the consultants, and the in-house team—should be in agreement with the contents of the plan.

How often should the executive committee monitor the project? The answer is, "It depends." If the company has really assigned its best personnel to the job, then there will be a natural monitoring of the project's daily activities. The company professional assigned to serve as the project owner or sponsor is in an ideal position to evaluate how things are going. Because company-wide SCM implementation projects last for several weeks or months, it is quite adequate for the executive committee to meet once a week or once in two weeks (with a provision to hold emergency meetings when necessary).

Another choice is to set up milestones in the project plan and have a meeting when the milestone's planned completion date is over. However, there are no hard and fast rules regarding how frequent the executive committee should meet. During the final stages of the project, when the system is being tested, the committee might need to meet more frequently to discuss the various issues that could arise.

It is the task of the project management team to report to the executive committee and present the facts and figures. Because these meetings are managerial in nature, the project management team should prepare a presentation that describes the situation at a level of detail appropriate to the audience. Very technical topics should be condensed, and excessive use of jargon should be avoided. It is a good idea for the material used in the presentation to serve also as documentation of the status of the project to that date. This is important, because, in order to track the progress at future meetings, it may become necessary to recall the issues presented in a previous meeting, so as to explain why the evolution of the work has taken a particular route.

Another objective of the executive committee meetings is to address the issues that involve decisions by top management. Such decisions will not be made at every meeting, but when they have to be made, they need careful preparation. The project management team should circulate details about the issues well in advance so that the committee members can do their homework and come prepared for the discussions. It is the duty of the project management team to analyze alternative solutions and their advantages, disadvantages, and consequences and circulate them to the committee members well in advance of the meeting.

So in an SCM implementation project, the work plan or the project plan is of paramount importance. Adherence to the plan, along with constant monitoring and the taking of appropriate corrective actions before the project gets out of control, will ensure the success of the project. The key players in project tracking and monitoring are the project management team and the executive committee.

To guarantee that a complex and sophisticated project that requires technological and cultural changes in the company will reach its conclusion successfully, it is not enough to sell it or approve it. The project team needs to resell it constantly, by demonstrating that it is evolving in an appropriate manner toward the stage at which benefits will be generated as initially anticipated.

Problem Resolution

During the implementation, many issues will be raised and require resolution. The danger is that some of these issues, having been identified, will be forgotten only to surface at a later date, perhaps after the system is live. Thus, ideally, there should be an agreed procedure for recording issues and their resolution. Whether this is by use of a flip chart or a more sophisticated process, it should be accepted practice that when an issue is raised, it is recorded. When the issue has been resolved, it can then be marked as closed. By adopting a simple approach, unresolved issues are highlighted. This may reveal that some issues have simply been ignored for the time being. Alternatively, it may reveal issues that need additional support, perhaps an executive decision or vendor support. By keeping track of problems, they can be dealt with systematically, thereby reducing the likelihood of something unpleasant manifesting at a later date.

System Issues

Although much of the attention focuses upon the upfront activities of the implementation, in the background there is a lot of work dealing with the technical issues. Tasks include the installation and commissioning of both hardware and software. These tasks will be identified on the project plan, but those involved will normally be the IT systems personnel. Technical support will be provided, as required, by the appropriate vendors. From an SCM perspective, there are a few questions that should not be overlooked:

- How does the system perform when the SCM application is under heavy use?
- How quickly will storage space be consumed when the system is live?
- What is the backup procedure?
- Can the live domain be duplicated so that work can be done in the other domains, such as new process development, without affecting the live domain?
- How many alternative domains can be created? How long will the transfer take?
- What happens when two people try to access the same data?
- Does the system lock, and if so how is it unlocked?
- Is it necessary to change the locations of PCs and printers?
- If the intention is to use preprinted stationery on dedicated printers, can this be done, and if so what is involved and when is the time to do it?
- How secure is the system?
- How will user access be selectively restricted? Is it by screen or by field?
- How are passwords managed?
- What user menus need to be generated, and how will this be handled?
- Is there an automatic logout facility if an account is logged-in and not used for a period of time?
- What is the disaster recovery procedure?

This is not an exhaustive list, but it reveals the diversity of issues that need to be addressed. The earlier these issues are identified, the more time there is for dealing with them. A minor detail like a dedicated printer for preprinted stationery may prevent purchase orders from being issued if the printer cannot be configured so that the required data prints correctly. In a live situation, the delay involved in finding a solution as to how purchase orders will be printed if the printer cannot be configured could result in operational stoppages due to materials not being available. Likewise, users need to be issued passwords and trained in the logon procedure. Issues like what happens if users forget their passwords should be addressed and solved.

Consultants

Business consultants are professionals who specialize in developing techniques and methodologies for dealing with the implementation and the various problems that will crop up during the implementation. They are experts in the administration, management, and control of these types of projects. Each will have many years of implementation experience with various industries and have knowledge of time-tested methodologies and business practices that will ensure successful implementation. They will be good at all phases of the implementation life cycle right from package evaluation to end-user training. The only problem is that they are expensive—very expensive.

There are many myths and ambiguities about the consultants and the role they play in an SCM implementation. Several of these myths are listed—and refuted—as follows.

- *Employing a consultant or consulting firm can make the implementation successful.* This is not true as the success of the SCM implementation depends on top management support, good planning, the right package selection, an excellent implantation team, and a host of other factors. A consultant can help educate you and point you in the right direction, but the success of the SCM implementation project is within your hands, and employing a consultant cannot make an implementation successful.
- *Consultants should lead the project.* Consultants are hired for their expertise, and they should function in an advisory capacity. The person leading the project should also be the project owner, one who will stay with the organization long after the implementation is complete. Making consultants project leaders is a recipe for disaster as they will go to another company after implementation.
- *Consultants know everything.* Yes, consultants are hired for their knowledge and expertise. However, assuming that they know everything is wishful thinking. Consultants do not know how every company works or its business details. Also, the experience and expertise, and hence knowledge, of consultants differ. Accordingly, ensure that employees learn from the consultants whatever they can—knowledge transfer. Once the employees become knowledgeable, consultants become redundant.

- *Having more consultants will increases the chances of success*. In fact, having too many consultants is a sure recipe for failure. Too many consultants can create too many views and opinions and can slow down the decision-making and reaction times. So, limit the number of consultants, but make sure the ones you have are the best you can afford.
- *Consultants who are expensive and who have fancy degrees are better.* Consultants should be judged by their reputation in the field—how they are regarded by other experts, how many projects they have consulted for, and how varied their experience. Consultancy fees and degrees are not at all yardsticks of excellence.

Consultants provide a wide variety of functions, often filling in gaps. Some of the positions that consultants can fill include project manager, team leader, team member, service representative, and end user. A consultant's success depends upon a number of factors including computer literacy, conceptual skills, software knowledge, industry knowledge, maturity, problem-solving capability, communication skills, and organizational skills.

The success of any particular consultant can vary tremendously from company to company and from situation to situation. Surprisingly (to some), a consultant's industry and software knowledge does not correlate strongly with his or her success or capability to help a company. Case after case has identified consultants lacking in software and industry knowledge who were, nevertheless, able to consistently outperform other consultants considered the most knowledgeable in software and industry. These consultants show strong interpersonal communication skills, are self-starters requiring little or no training, and have good computer literacy, problem-solving capability, and conceptual skills.

Choosing a consultant or consultants is an important decision that can affect the success of the SCM implementation. It should be done with great care as the right consultants can steer the SCM implementation clear of pitfalls and lead it to on-time completion. Before choosing a consultant, the project team should consider the following:

- *Experience:* Do they have experience in the organization's particular industry?
- *Success rate:* How many successful implementations have they completed?
- *Size and scope:* What are the sizes of the companies and number of users involved in their previous implementations? What was the scope of those implementations? Were they big-bang or phased?
- *Continuity:* Do the consultants provide ongoing service and support? What is their annual retainer for support after implementation? What are the terms and conditions for such an arrangement?
- *Training:* How good is the knowledge transfer? Will the consultants train company employees so that they can implement the SCM system in other offices or branches of the organization? Do the consultants train employees to operate and maintain the system independently?

Many of the big consulting firms, having forecasted the SCM boom, invested a great deal of money in developing a range of consulting services in this field and

assigned many of their professionals to become specialists in the various aspects of SCM packages and their implementation. These firms researched the various products, developed an in-depth understanding of each product's strengths and weaknesses, worked by the side of the SCM vendors, confirmed that the vendor's package worked, learned the tricks and techniques of the trade, determined the pitfalls and mistakes that should be avoided, and thus created a pool of experts who could handle the SCM implementation without failure.

Thus, consultants are people who have made the business of SCM implementation their business and invested huge amounts of money and manpower toward that purpose. So when you want to obtain the services of these consultants, the first question you should ask is, "Are they going to be expensive?" The answer is a definite YES. The consultants will be expensive, so the company will have to formulate a plan to make the best use of the money spent on consultants. If we study the statistics, we can see that a well-selected, integrated system that is successfully implemented and successfully working usually pays for itself in a relatively short period—between 10 and 30 months. If you analyze the cost breakdown, you will find that the most expensive part of the implementation was the consultation charges. For a typical SCM implementation, the cost of consultants is 1.5 to 3 times every dollar invested in the software product. This sounds amazing, but it is true and it is also true that the software will pay for itself—the software cost, the consultant's charges, and other expenses incurred during implementation—in the above-mentioned period (10–30 months). However, the catch is that the product has to be the right one, and the implementation has to be successful. That is why the expertise of the consultants becomes invaluable, and the money spent on good consultation is never wasted. So finding the right consultants—people who have the necessary know-how and who will work well with the company personnel, those who will transfer their knowledge to the company's employees and who are available in case their services are required again—is very important.

Consultants provide three general categories of services—management, application, and technical. Service providers divide consultants into these three general categories. Some consultants can perform two of these categories, but it is rare to find consultants who can perform all three.

Management consultants focus primarily on the function of management as it relates to the organization of resources and business process flows. Management consultants often participate in project management and provide high-level direction for the overall successful implementation and use of an SCM system.

Application consultants focus on the process of communicating, teaching, demonstrating, and configuring software for the business process flows. A management consultant may consult on how to perform a business process flow, whereas an application consultant would show the company how to perform the business process flow in the new software.

Technical consultants deal with technical issues such as database conversions, source code modifications, communication protocols, operating systems, software installation, hardware systems, and integration programs. Technical consultants work closely with application and management consultants.

It can be difficult to estimate the proper ratio of management, application, and technical consultants required for a particular SCM project. Even slight changes in

the SCM implementation strategy can affect the ratios. The ratios fluctuate between the beginning and the end of an implementation.

Role of the Consultants

The role of the consultants is very familiar to all of us because we have seen many of them in action. The company places its trust in the consultants for the achievement of its business objectives. In fact, it is a better practice that the contract between the company and the consultants should have all the performance clauses in place. The consultants should guarantee the success of the project and should be able to show results (quantifiable results like reduction in cycle time, increased response time, and improved productivity) to the satisfaction of company management.

Consultants are responsible for administering each of the phases of the implementation so that the required activities occur at the scheduled time, at the desired level of quality, and with the effective participation of all those who must participate. For keeping the promises that the consultants have made during the negotiations, they have to transform their approaches and methodologies into detailed work plans. Methodologies will have to be converted into tasks and should be allocated to the right people. The time schedule for each phase and each task has to be determined, and the project plan has to be finalized.

Consultants should add value to the project. They bring know-how about the package and about implementation—know-how that is not included in the standard documentation. This know-how (also known as practical knowledge) is derived from their expertise, which stems from practical experience. Because the consultants have seen many projects and have made or seen many mistakes, they can avoid the phenomenon of "reinventing the wheel." They will know what will work and what will not. Thus, by eliminating the trial-and-error method of implementation and doing it right the first time the consultants help to save huge amounts of money, time, and effort.

Consultants should also know how to remain impartial while questioning current company processes in an effort to promote better businesses practices and better implementation results. They should strive to improve the company's business processes so that the software package can be used as it was originally intended to, by its developers. Refining the company's processes can only optimize the performance of the system and maximize future user satisfaction. Consultants are also responsible for analyzing and clearly addressing customization issues. They must be able to distinguish between the "must-have" and "nice-to-have" items and decide on the level of customization. This is an area where consultants have to use their diplomatic skills, as company personnel might want to customize all the aspects. It is the duty of the consultants to present the advantages and drawbacks of each area and reach a consensus decision, which should also be the right one. Consultants need to position themselves in such a way as to balance their loyalty to the client and the project, with that of defending the package vendor, when such a defense is technically correct. This is indeed a very difficult job (like a tightrope walk), and that is why consultants are paid such huge amounts for their services.

It is the duty of the consultant to understand the total context and scope of the envisioned work and to know when to alert company management about actions

and decisions that must be undertaken so that the job will not be compromised, and the implementation will not be jeopardized. Maintaining technical documentation on the project also falls within the duty of the consultant. Consultants will leave once the project is complete, but the knowledge of the project cannot depart with them. So consultants should create a knowledge base and train enough people so that the work they have started is continued.

Contract with the Consultants

Consultants are another group of people who play a vital role in the SCM implementation project. Compared to the package, the consulting services are of a more subjective nature. It is difficult to measure whether the company is getting its money's worth from the consultants. The company's objective is to make the package work successfully as documented in the vendor's manuals. What the company expects from the consultants is that they will make the project a success, that the implementation will be completed without time and cost overruns, that the user training will be done to everybody's satisfaction, and that the consultants will train a group of people (and transfer their knowledge) so that when they leave there will be enough employees in the company who can carry on the work.

Even though the consulting services are subjective in nature, the company can include performance and penalty clauses in the contract. For example, the company and the consultants can agree upon the completion date and the implementation budget and the projected improvements (e.g., x% increase in productivity and y% reduction in response times) and then include these in the contract. Then, the consultants can be held accountable. The following are some of the points that should be included in the contract with the consultants:

- The profile of the consultant's team with resume of each member;
- The consulting fee and payment conditions;
- The time schedule and implementation budget;
- The projected improvements in quantifiable terms and the time required for showing the results;
- The implementation methodology;
- The terms and conditions of knowledge transfer and employee training;
- A list of deliverables (e.g., reports, manuals, and knowledge bases);
- Other specific activities the consultants are supposed to do;
- The reporting mechanism to the company management;
- Project monitoring and status reporting systems.

Package Vendors

Vendors are the people who have developed the SCM packages. They are the people who have invested huge amounts of time and effort in research and development to create the packaged solutions. If one studies the history of SCM packages and finds out how each package evolved, then it soon becomes evident that every SCM package grew out of the experience or opportunity of a group of people working in

a specific business who created systems that could deal with certain business segments. With the SCM marketplace becoming crowded with more and more players entering the market and competition increasing, today's SCM packages have features and functionality to cater to the needs of businesses in almost all sectors. SCM vendors spend crores of rupees in research and come up with innovations that make the packages more efficient and flexible and easier to implement and use. Also, with the evolution of new technologies, vendors constantly have to upgrade their products to use the best and latest advancements in technology.

Choosing the right software vendor goes beyond evaluating software functionality. There has been a gradual movement among a handful of the largest software vendors to take a one-size-fits-all posture. Some vendors believe functionality ratings are no longer important since all software systems are beginning to look alike. While appearances may tend to support this theory, reality paints a different picture. Merely having a particular function does not guarantee that users will be able to capably work with it. If two vendors offer a function required in a specific industry segment, and one specializes in deploying it in that segment while the other does not, the difference can be dramatic. The one-size-fits-all garment may fit everybody, but does it look good on everyone?

Vendor selection is not a popularity contest, and bigger does not always mean better. While the financial stability, ensured longevity, and broad spectrum of offerings provided by the top vendors are good reasons for selecting them, size is not without its downside. Size breeds bureaucracy, and bureaucracy hampers personal attention and agility. While smaller vendors that are not quite household names may carry increased risks in the area of long-term longevity, they may actually provide a better solution if they specialize in your industry segment rather than covering a broad spectrum of industries.

You have the greatest leverage with your vendor once you have made the decision to buy its software but have not yet issued the purchase order. Waiting for the end of their quarter can help you get the best price. Also, view the financial stability of your future relationship as important—you will receive positive support from your vendor as long as it is profitable for the vendor to do so. The relationship is a balancing act. Interest your vendor in getting the job done right, on-time, and within budget, but watch out for penalties that may increase project pressure and sour the relationship.

It is important to remember that vendors, as long as they provide working software and capable personnel, really have very little responsibility for your overall success. Responsibility for success of failure lies within the four walls of your business, and if you import failure in the form of a third party, it's still your responsibility.

Vendors and Vendor Management

Nowadays most SCM systems use some sort of tool. (In other words, manual SCM systems are very rare). So the implementation team should include vendor representatives. Because the vendors are the people who know the tool best, they definitely have a role to play in the implementation. The vendor should supply the product and its documentation as soon as the contract is signed. Once the contract has been exchanged, the vendor will guide the company, in particular the project

manager and his or her team, through a series of events culminating in the use of the tool. The project manager, as with all the other resources, will need to manage the vendor so that everything progresses as intended.

One cautionary word concerns the power of the vendor. The vendor potentially has the upper hand in the client-vendor relationship. The client, having signed the contract and perhaps having given a preliminary payment, will be reluctant to terminate the relationship should the vendor fail to meet expectations. Consultants may not be available due to split commitments. Software bugs may not be fixed when required. Software links may simply not work. These all contribute to delay. It is the project manager's task to manage this.

The vendor will most likely appoint a single point of contact, the vendor's project manager. The role of this person will be to provide support to the project manager, advise upon and agree to the project plan, manage the client account, and coordinate vendor-provided resources. A procedure should be agreed on between the two managers about how work done by the vendor is to be authorized by the client. The vendor's project manager will be the first point of contact for resolution of problems whether they are technical, best practice–related, or pertaining to vendor invoices. Within the contract, there should be a clause defining how problems are to be handled and the timescale allowed. If problems are not resolved, then an escalation path should be defined identifying the people to be contacted. Communication should be backed up in writing. This reduces the likelihood of misunderstandings.

Only after the software is delivered can the company develop the training and testing environment for the implementation team. The vendor is responsible for fixing any problems that the implementation team encounters in the software. So the vendor should have a liaison officer who constantly interacts with the implementation team. Another role the vendor has to play is that of the trainer—to provide the initial training for the company's key users, people who will play lead roles in the implementation of the system. These key users are the ones who will define, together with the consultants (external experts), how the software is to serve the company. These in-house experts will decide how the functionalities are to be implemented, as well as how to use or adapt the product to suit the company's unique requirements. So it is critical that these key users be given thorough training on the features of the package. Vendor training should show key users how the package works, what the major components are, how the data and information flow across the system, what is flexible and what is not, what can be configured and what cannot, what can be customized and what should not, the limitations, the strengths, and weaknesses.

The objective of vendor training is to show how the system works, not how it should be implemented. This means that the vendor demonstrates the product as it exists and highlights the available options. The employees participating in the vendor training should try to understand the characteristics of the package and the impact of the system on the company's business processes. The trainees should use these training sessions to question the vendor on all aspects of the system.

The external consultants (or the package or SCM experts) also have a role to play during this vendor training. They should participate in the training sessions to evaluate how the users react to the reality that is starting to take shape from the detailed presentations and demos. Consultants should also ask questions that the vendors are trying to avoid and the users are unaware of. This is the best way

to present the real picture to the users, and it will also prevent the vendors from making false claims.

The project manager should monitor and control the costs incurred by the vendor. By monitoring the amount of time that the vendor's personnel are on-site, it is possible to monitor consultancy costs, a potential area for overspending. Mistakes do happen, and it is advisable that vendor's invoices are checked. Queries should be brought to the vendor's attention for resolution, for which there should be a provision within the contract regarding the withholding of payment. Finally, the project manager should ensure that the vendor is paid and on-time.

Role of the Vendor

First and foremost, the vendor should supply the product and its documentation as soon as the contract is signed. Only after the software is delivered, can the company develop the training and testing environment for the implementation team. The vendor is responsible for fixing any problems in the software that the implementation team encounters. So the vendor should have a liaison officer who should constantly interact with the implementation team.

Another role the vendor has to play is that of the trainer—to provide the initial training for the company's key users, people who will play lead roles in the implementation of the system. These key users are the ones who will define, together with the consultants, how the software is to serve the company. In other words, it is these in-house functional experts who will decide how the functionalities are to be implemented, as well as how to use or adapt the product to suit the company's unique requirements. So it is critical that these key users are thoroughly trained on the features of the package. Vendor training should achieve the goal of showing key users such information as how the package works and what its major components are, how the data and information flows across the system, what is flexible and what is not, what can be configured and what cannot, what can be customized and what should not, the limitations, and the strengths and weaknesses.

Now some of you might ask, "We are hiring consultants who are experts in the package, so why can't we get training from the consultants?" This is true. Most consultants are capable of providing sound training for the packages. However, we are hiring the consultants to implement the system. The objective of the vendor training is to show how the system works, not to show how it should be implemented. This means that the vendor demonstrates the product as it exists and highlights the possible options available. The employees participating in the vendor training should try to understand the characteristics of the package and the impact of the system on their business processes. The trainees should use these training sessions to question the vendor on all aspects of the system.

The consultants also have a role to play during this vendor training. They should participate in the training sessions to evaluate how users react to the reality that is starting to take shape from the detailed presentations and demos. Consultants should also ask questions that the vendors are trying to avoid and of which the users are unaware. This is the best way to present the real picture to the users, and it will prevent vendors from making false claims.

The role of the package vendor does not end with the training. The vendor also plays an important project support function and must exercise quality control with respect to how the product is implemented. It is the vendor who understands the finer details and subtleties of the product and who can make valuable suggestions and improvements that could improve the performance of the system. It is also in the best interests of the vendor that this participation continues, because if the implementation fails, most of the blame will fall on the vendor. In addition, a successful implementation means another satisfied client, improved goodwill, and good referrals. So the vendor will continue to participate in all the phases of the implementation mostly in an advisory capacity, addressing specific technical questions about the product and technology.

The vendor has other responsibilities, also. There will be "gaps" between the package and the actual business processes. The software might have to be customized to suit the company's needs. Customizing means altering the product so that it is suited for the company's purposes. The choice of whether to customize or not is one that can have enormous impact on the project, and it often constitutes a point of conflict between consultants and users. However, if the decision to customize has been taken, it is the vendor's duty to carry out the necessary modifications. This is because only the vendor knows the product well enough to make the necessary changes without affecting the other parts. Moreover, the company must get a guarantee (in writing) from the vendor that despite the customization, it will benefit from the future software improvements introduced by the vendor.

Contract with the Vendor

Most software vendors (for that matter all the people involved except the employees) will have standard contracts drafted by their legal department that include clauses to safeguard their interests. The contracting company's legal department should go through the contract, and if it finds the terms and conditions agreeable, sign the contract.

In reality, however, the contract signing process is not this smooth. Often, it will require lawyers from both sides to sit together and iron out differences. The package vendor will have copyright clauses in the contract. As the creator of the package, it is their intellectual property. So any abuse or misuse of the package by the contracting company will be a violation of the copyright. Sometimes, the contracting company might want access to the source code and the design documents, so that they can make the required modifications—customization—to the source code. However, package vendors usually will not want this. The main reason for this reluctance is that once the package vendor gives the source code and design documents, it is very easy for anybody who has access to these documents to misuse it.

Also, it is better to give the task of customizing the package—by modifying the source code—to the package vendors themselves. The vendor has developed the package and will have a better knowledge of the source code. The vendor will have the complete knowledge of the impact of a change to the entire system. Another point that should be remembered is that the relationship between the SCM vendor and the contracting company is not a one-time affair. The vendor should be made a

partner in the company's future plans. The vendor will be upgrading its product as the technology advances, adding new modules and features. The company should get the benefit of these upgrades. Being an existing customer, the company should get preferential treatment. The contracting company should include clauses to make these things happen. One problem of the company modifying the source code is that it might prevent the contracting company from getting support on upgrades and future versions. So, it is best to leave the customization to the vendor. However, if the contracting company is not sure about the stability of the vendor and its continued existence, then the contract can include clauses to keep a copy of the source code and the design documents with a third party (like a bank) and release it if the vendor disappears from the scene or stops support for the product. This agreement will be agreeable to the vendors also.

So the contract with the vendor should address—in addition to issues about source code and modifications—the following points:

- Value of the software and conditions of payment;
- List of deliverables (e.g., software and documents);
- Mode of delivery and installation help;
- Copyright and ownership issues;
- Software licenses;
- Third-party software compatibility, integration or interfacing, and integration support;
- Operating system;
- Hardware or liability;
- Conditions and concessions for acquiring complementary modules in the future or for increasing the number of end users;
- Cost of implementation training;
- Cost of end-user training;
- Annual maintenance fee;
- Warranty or guarantee terms;
- Terms and conditions for the receipt of new versions or upgrades;
- Details of technical support (e.g., on-site or telephonic);
- Terms and conditions for customization;
- The profile of the vendor's team who will be assisting the company in implementation;
- Other specific responsibilities assigned to the vendor;
- Cancellation of license.

The above is by no means a comprehensive list. You will have to add clauses that are specific to your case. The objective is to protect the company's interests and whatever is needed for this protection should be in the contract. Obviously, the content should be rigorously checked and the requisite amendments and additions made. Particular attention should be paid to the meaning of the words used, in particular the subtle differences of specific words. The requirements definition and subsequent amendments should be included. The content should take into account all eventualities for the lifetime of both the software implementation and its license. Things to watch out for include the following:

- Payment profile. One possible consideration is to schedule payments in line with the attainment of targets and deadline.
- License fee based on named users or number of concurrent users.
- Contingencies for when a software bug prevents the implementation of a specific functionality and the bug fix will only be available in the new upgrade available at a later date?
- The process for controlling software modifications and their testing and the warranty period?
- Who owns software customizations and the associated intellectual rights?
- Who has responsibility for sourcing new hardware, and what happens if it can be sourced cheaper elsewhere?
- Cost of transferring software from old to new hardware and who will bear them?
- Restrictions on who can use the software (e.g., satellite operations);
- Entitlements for additional service;
- Response procedure for problems including escalation route;
- Escrow arrangements should the vendor go into receivership.

These are just a few pointers. In view of the size of the investment, the safest recourse is to seek legal expertise. Indeed, it may be prudent not to dismiss the second choice of vendor should negotiations breakdown.

Training and Education

SCM implementation is primarily a people project. A majority of the people problems like employee resistance, noncooperation, and improper use of the SCM system. are the result of ignorance, fear of failure, and fear about the future. Most experts and SCM implementation veterans agree that the best way to combat this issue is through comprehensive training and education of the users of the SCM system.

Training is perhaps the most misjudged activity of the implementation life cycle. A major complaint is that not enough training is done. One of the most common mistakes of all SCM implementations is underestimating the time and cost of training end users. Although training is a project-managed activity, it appears to be widely neglected or is inconsistent in application. Furthermore, since the greater part of training takes place toward the end of the implementation cycle, when it looks like overall costs will exceed budget, training is the first activity to be curtailed.

The major pieces of the SCM training process are described as follows.

- *Planning:* Identifying the elements needed to structure the training direction.
- *Budgeting:* Determining the investment required to create the infrastructure and impart the training.
- *Staffing:* Determining resources (internal and external) and needed prerequisite skills.
- *Partnering with the business:* Developing a shared responsibility and success plan.
- *Organizational issues:* Red flags to look for that could impact the plan and coping strategies.

- *Curriculum development:* Discussing best practices for SCM projects.
- *Implementation:* Rolling out the training.

Overview of Training

This section provides an overview of the main issues to be considered when embarking upon the training activity. Although it may be convenient to take an informal approach to training whereby people "pick up knowledge and skills as they go along," this is unpredictable in terms of a successful learning outcome. A more formal approach to training tends to involve the following stages:

- *Defining learning objectives:* What will the learner be able to do as a result of the training?
- *Determining content:* What skills and knowledge are to be developed?
- *Planning:* When and how will the training be delivered? What resources, materials, and facilities are required? How will the content be structured?
- *Delivery:* The experience of the learner.
- *Assessing learners:* Have the learners met the objectives?
- *Reviewing effectiveness of the training session:* What went wrong? What can be done better?

Those at the receiving end of the training are initially the project team members and the system administrators and then later in the project they are such personnel as the developers, QA team, testers, project leaders, and managers. Each group of learners will have different requirements. Thus, the nature of the training is likely to be different for each different stream of learners.

Training that occurs too far in advance of the go-live date will likely be forgotten. Training that occurs too late will not be done in time and can lengthen the stabilization period. In order to fit training between the current time and the go-live date, the firm must consider the amount of time in that gap and the amount of time required by the users to learn.

The amount of training required is a function of the particular module for which users are being trained. In some cases, it can take up to six months for users to get comfortable and proficient with the SCM software.

Time spent on training is time not spent on day-to-day activities. Not surprisingly, there have been a number of different solutions used to ensure that workers get enough time off for training. Employees can put in extra hours to accommodate training hours. Still other firms have made use of temporary employees.

The hours assigned to training signal how important it is to the implementation. Training scheduled during working hours indicates its importance, whereas training scheduled outside of working hours suggests that training is not as important as day-to-day responsibilities.

The most pressing activities often get the most attention, to the detriment of other, less pressing activities. This means that productive work usually receives greater attention than training activities and that user SCM training might be pushed aside if firms do not ensure that users take it seriously. As a result, firms

have introduced different penalties and incentives. In some companies, training is deemed mandatory for certain critical users.

Training end users on how to use an SCM system is a mix of technology, processes, and domain-area content in order to provide a context for the system. Otherwise, training can become a useless exercise. It is always better to train on the concepts first and then show the end users how to use the system. When the end users have a good idea about why things are done, they will learn the tasks required to accomplish those processes faster.

Most training programs come from an analysis of what the specific company has and what it needs to accomplish. Almost all SCM approaches to training have an element of classroom training. However, other formats used include training over the Internet, computer-based training, and self-study. Based on conversations with consultants, one approach that is consistently well accepted involves designating a member (or group of members) of the client organization as "super users or champions," who can then be responsible for training others. Training super users has a number of advantages. First, this approach has been found to facilitate buy-in from users, because those who supply the training are people the users know. Second, the existence of super users shows other users that learning about the system is important. Third, developing super users develops an important understanding at the user level.

Because SCM engagements are often behind schedule, firms may try to speed up their training. The ability of firms to speed up training depends on the firm's needs, personnel, and previous training. Some personnel are likely to be quicker learners or are able to spend more time on training than other personnel. The training will likely be faster to the extent that the new system is similar to the old system. However, trying to speed up the training is potentially dangerous, with a high risk of failure.

Training Costs

Training costs will vary across SCM implementations. The training budget can be 15–20% of the total project budget. Training is one of the most important hidden costs of an SCM implementation. In most cases, there will always be training cost overruns. So there should be a provision to allocate addition budgets for training when required. The organization can reduce the training costs by first training a batch of employees and then making them trainers so that they can train their colleagues. This training will be more effective as people will be more receptive to ideas and more open about their own ideas, fears, and concerns when the training is conducted by a person they know. Some of the items that contribute to training costs are described as follows:

- *Training planners and content developers:* For training to be done properly, it has to be planned well, and for this the training needs of the employees must be identified. Once the needs are identified, the content that will address those needs have to be developed. The people who do this—external SCM training consultants and internal experts—have to be paid.

- *Hardware infrastructure:* To create a training center, the training team will have to spend money on infrastructure like computers, training aids, and stationery and also on classrooms and labs to facilitate effective and uninterrupted training.
- *Software:* This includes the demo or training versions of the SCM software that is being implemented and other software packages like word processors, spreadsheets, presentation software, e-mail programs, and Web browsers that are necessary to make employees computer-savvy and proficient in the SCM package the organization is implementing. All the employees need not be given training in all the SCM modules; after an overview of how the different modules work and how they are interrelated, employees can be given comprehensive training on the SCM module that he or she will be working and also on the necessary software package that he or she will have to use for effective communication.
- *Trainers:* These include the salary of the trainers—both in-house experts and external consultants. One way to minimize the cost of trainers is to train a batch of employees as trainers.
- *Vendor consultants and training materials:* The vendor should provide the training materials and training consultants for the initial phases—until the in-house trainers are being trained and are capable of conducting training on their own. The vendor should also provide updated training materials and conduct refresher courses for the in-house trainers as and when upgrades and changes are done to the SCM system. These issues and associated costs should be negotiated during the package selection and should be part of the SCM budget.
- *Support staff:* The cost of people who do the various administrative tasks should also be accounted for in the training budget. These people are important as they are the ones who have to ensure that all the people are trained and that the training goes on smoothly. Their tasks include scheduling employees into appropriate courses, tracking completion, scheduling make-up courses, and adjusting training schedules. They are the ones who should contact the vendor consultants and other SCM consultants when the need arises and when the in-house trainers need additional training.

The above list by no means is an exhaustive one. As discussed previously, the training costs will depend on such factors as the organizational culture, current knowledge, and existing infrastructure. So the amount of money that needs to be spent on each of the above items and others will vary among organizations.

Need and Importance of Training

We have seen that many SCM implementations fail and that many more fail to deliver the promised results. What is behind SCM disasters? Most implementation experts have pointed out that the main culprit of failed SCM implementations is the end-user training. Experts say that the technical training of the core team of people who are installing the software is done properly and there is no problem

in that aspect. The problem lies in the education of the broad user community of managers and employees, who are supposed to actually run the software development with it. This training is not done properly, resulting in the wrong and improper usage of the system.

Many studies have revealed the fact that as few as 10–15% of SCM implementations have a smooth introduction that delivers the anticipated benefits. The remaining firms either experience teething problems or a significant shortfall in delivered benefits, and the difference between the successful 10–15% and the rest is better training.

Everyone knows that training is important, especially SCM software vendors. They earn handy revenues from "design once, recycle many times" training courses. The third-party training firms, who conduct courses on how to operate an SCM vendor's system, also know the importance of training. The variety of training formats available is amazing—on-site training, Web-based virtual classrooms, computer-based training, knowledge warehouses, video courses, self-study books, context sensitive help screens—an almost endless menu to suit almost every need and budget.

SCM training has become a giant business in its own right, and the training business is expected to grow exponentially. The logic is inexorable—the better the training, the faster you will see the business metrics move in the direction you are looking for. However, the problem with such training is that it is not good enough. It is just a walkthrough of the system and teaches users what to do. This will not help users to understand what they are doing and why.

The education should impart to users the ability to figure out the underlying flow of information through the business itself. The program should explain such factors as SCM basics, the business processes, how the SCM system functions, how it automates the business processes, and how the action of a user affects the entire organization. The focus should shift from mere training to providing education—with greater emphasis on education. Education will tell the users why they are doing it and help to win support for the project as it will enlighten the users. Training, on the other hand, will only tell them what to do and how to do it. There is a tendency for companies to fall into the trap of putting employees through training programs that are too software-specific. This kind of a training program will ignore the fact that SCM systems are designed to operate by codifying a set of business processes.

Another problem is that training typically occurs at the end of the implementation cycle, when activities are often running late and being compressed. So training, too, gets squeezed in as a last-minute activity. Accordingly, it is important to time the start training programs so that they will be nearly over when the system goes live and then continue for a few more sessions to clear the issues faced during the actual interaction with the SCM system. One of the results of not providing a proper training program is that users fail to appreciate the consequences of their actions, often with disastrous results. Informal practices that worked fine in the era of paper procedures or stand-alone legacy systems can have catastrophic effects on an integrated SCM environment.

Training in how to operate the system will not, however, help the middle manager see far enough down the road to decide to forgo the short-term benefit of shipping

product "come what may." Only a broader based, holistic education in the company's SCM-mediated business processes will do that. If end-user and middle-management training is so important, why it is not given more priority? Companies have begun to wake up to the fact that training is a key requirement.

Training Phases

The training strategy should include two phases of training—one before implementation and the other during and after implementation. The implementation of the project commences with the training of the project team so that members are able to carry out their tasks. During the implementation and after the implementation, end users are trained on SCM basics, process changes, and how to use the SCM system.

Preimplementation Training

Implementation of the first phase of the training strategy is the training activity that relates to the training of the project team and the system administrators. For the project team, the focus of the training will be understanding the functionality of the software. Training on such subjects as best practices, process mapping, training skills, and documentation may be provided by the vendor, but this will vary from vendor to vendor. A local higher educational establishment or other training organization may be able to fulfill any gaps. The training of the system administrators will focus on technical aspects of system installation, maintenance, report writing, and any other identified issues.

The objective of the training is to transfer knowledge and skills about the application, implementation practices, and operational best practices from the external trainers to designated internal personnel. Whilst most of this will be done in more formal proceedings, the transfer of knowledge about the software functionality tends to be done on a more informal basis. However, since this transfer of knowledge about the functionality need not be effectual, it is worth examining this specific area more closely.

Content

Team members' understanding of application functionality is critical for the effective development and introduction of new processes. Without it, it becomes impossible to make the most of what is an expensive investment. While the presales demonstrations will promote the merits of the functionality of the software, its drawbacks may be withheld.

The first true exposure to the software comes when it becomes necessary to make the functionality do what is required of it. There are various levels of knowledge and skills required by the project team to develop business processes that utilize the software. Each member requires knowledge about how to navigate around the system and the detail of the functionality of concern.

A content matrix is the most useful reference for training developers, and it will grow into a huge, complex document over the life of the project. Simply put, the

matrix lists the skills that should be imparted and tasks that must be taught and information to be presented for each job role in the new environment. In its final form, it will not only guide development but also provide a completion checklist. Another useful document, using the curriculum matrix as its basis, is an employee job map that will map employees in each department to appropriate job roles and necessary training required, giving a departmental and end-user snapshot of the training needed.

The vendor's consultants should be the experts on how the software functions. It must be expected that these consultants will sit with team members, going though the screens and fields for the processes of interest. This training mode tends to be hands-on. This is an ideal opportunity for the team members to learn. In time, each person should build up a good understanding about specific parts of the application. Their aim is to be able to experiment with different ways of using the software and to transfer this knowledge to others. These key users are responsible for ensuring that they learn as much as they can about the functionality.

If there is concern about whether project team members are developing the required knowledge and skills, then it may be desirable to assess them. How this is done should be established in consultation with the vendor, as that is where the expertise resides. However, gaps in a person's knowledge and skills will become readily apparent as he or she starts to try and work with the application independently. It is expected that this person will attempt to fill these gaps as they reveal themselves at the earliest opportunity.

Planning

Training planning normally begins during initial phases of the SCM implementation project, preferably during the project planning phase. It is during the project planning phase that the details of how to go about the implementation are decided. This is an excellent time to begin assessing organizational readiness and the current levels of end-user skill and knowledge in relation to those that will be needed in the new environment.

Many companies find that planning for SCM training is a multiphased process. The extent and detail of the project plan grows over several months, as companies uncover new facts and come to fully understand the scope of the initiative. There is no mystery to planning. All you really need to know is when to begin, the level of effort you need to apply, and the plan deliverables you need to develop. As the SCM project evolves, you can adjust and add detail to your plan and begin to schedule the work to be done.

The effort needed to perform the training assessment varies widely, depending upon project scope and how many people are affected. To do a thorough assessment of the training needs of an organization, it is better to form a team comprised of an experienced SCM training consultant and an experienced and senior person form the HR department of the organization.

The organization must allocate the time and budget to conduct the assessment, which will involve interviews with each end-user group and its manager, the executive team, selected SCM project team members, and support departments such as human resources, IT, and training.

An SCM training consultant can help the organization in constructing the questions and strategies for conducting the training needs assessment. After the assessment, the organization will have the following information:

- *Training plan for the end-users:* A detailed analysis of what they know (current skill set) and what they need to know (required skill set) and a training program to impart the skills that are lacking.
- *Pretraining orientation plan:* This plan should contain details and plans of the business skills and knowledge end users must obtain before SCM training begins, along with a time schedule for completing this training.
- *SCM training team training plan:* This plan should define the talent mix and resources required for the SCM training team. It should also identify the resources committed and needed for analysis, design, development, delivery, and administration of training. The plan should also point out the gaps between requirements and available talent and identify strategies for obtaining or developing resources.
- *Training delivery plan:* This plan should answer basic questions about how the curriculum is going to be developed and delivered. It should contain an overview of tools needed, logistical challenges, technical infrastructure requirements, training environment, and training delivery mechanisms. It should also provide an initial timeline for rollout of education and training.

Despite the informal nature of some of this training, it can all be planned. Dates can be established and people and locations organized. The emphasis of this phase is upon organizing and planning the training. External trainers carry out this delivery. The training is likely to comprise a mixture of formal workshops in a group environment and informal individual sessions. Most of the training will take place at the client's site in the project team room and will be delivered by the vendor.

More than one person should develop the necessary skills in each area as this reduces dependency upon that one person and prevents delays arising because of that person's other commitments. When completed, this training plan will become part of the project implementation plan and be monitored accordingly.

User Training (During and After Implementation)

The end user and managers are trained during implementation and after the implementation. The aim is to disseminate throughout the organization the project team's knowledge and skills relating to the application and the new processes. The expected outcome is the trainees being able to use the system.

Content

For this phase, a program can be developed that covers all the areas required. It can be organized into different themes to reflect different topics and audiences. While a general overview will appeal to everyone, the specialist areas will only be relevant to a limited number of people.

Some areas that will be relevant to everyone are SCM basics, business process, changed business procedures, automation of tasks by SCM, and fundamentals of computer usage like passwords, encryption, and security. Not to be overlooked is the fact that there may be new users who have no keyboard skills. If that is the case a touch-typing (typing without looking at the keyboard and searching for the keys) course can be organized. A better option will be to install touch-typing training software and ask the employees to learn it and pass the course.

Two audiences can be distinguished—end users and managers. While the former will be interested in how to use the system, managers will be more interested in how to get information from the system. Users can be further differentiated into casual users, normal users, and reflective users. The interest of casual users is limited to being able to perform certain tasks when required of them. Normal users are regular users of a specific suite of functions. Their main interest is using the system to do their job. Reflective users will want a deeper understanding of how the system works so that they can solve problems and make improvements. Thus, it may be appropriate to distinguish two levels of training: that essential to carry out tasks and a more detailed session on the finer points of the system.

The format of this training will tend to be structured into formal training workshops based around a PC. The data should ideally be that which they will be using when live so that real-life situations can be simulated. This may require data setup preparation. While the training material should be task-oriented, it will be explanatory in order to encourage an appreciation of why things are done in the way that they are. The project team room can be made available for trainees to practice on their own as a follow-up to the training courses. The trainers will be the new "experts" on the system, the project team members.

The timing of the training should be such that there is not a long gap between receiving training and using the application. A refresher course may need to be considered as a contingency. The cost of the training should be monitored against budget.

Planning

At the detailed level of each individual session, its preparation will be strengthened by its plan. The training plan should include the training objectives, trainer, trainer qualifications, audience, level and computer literacy of the audience, time, location, facilities required, content and content structure, method, resources or materials, and cost.

Training, Assessment, and Review

The training is complemented by an assessment to ensure that it has been successful and that the knowledge assimilation is satisfactory. During the training, the trainer will be confronted by a mixture of attitudes and expectations. While there will be those with a positive outlook, there will be others who have a negative view about the situation. Likewise, some may have high expectations about the quality of training and fail to appreciate the inexperience of the trainers. Thus, the trainers need to be aware that they may not be well received and be able to

respond accordingly. This highlights the importance of the trainers being trained in the training process.

Training should result in the trainees developing the requisite skills being taught. To ensure that this is being done, some form of assessment should be carried out. The emphasis is upon assessing what level of skill or knowledge has been attained. However, it should be remembered that the trainees are less likely to retain this knowledge or skill the longer the time that passes before they are required to use it. Thus, the assessment may only reveal what the person is capable of attaining rather than providing an indication that the person is competent at a specific task. It may be necessary to provide refresher courses closer to the go-live.

After each session has been completed, it should be reviewed to reflect upon the problems and to assess what worked and what could have been done better. The issues raised may be concerned with the material being presented and highlight the need for its revision. Alternatively, it may highlight issues with people and the need to establish a strategy for handling them. Each situation will be different, but each will contribute to an accumulating wealth of experience that, when continuously fed into subsequent training sessions, will make these sessions more successful learning experiences for the trainees.

Training Strategy

A training strategy can be developed defining the training policy and outlining the training program. Each stream will be identified and outlined in terms of the above stages. The strategy will provide an overview of the training objectives, identifying the people involved and the different streams and the content of each stream, organized into courses and sessions. A plan will provide an overview of where, when, and how the training will be delivered. Preliminary consideration will be given to the assessment of the learners. How can their knowledge and skill competencies be assessed? Furthermore, consideration is given to the effectiveness of the training and how this is assessed. Finally, the projected cost will be calculated. These costs can then be used to set a budget. The resultant strategy provides a framework within which to go about the training activity.

If the company accepts the strategy, it can be implemented. If the strategy is not accepted then it needs to be reviewed. A core issue is the company's commitment to training. The right balance needs to be struck between getting the training right and the training being cost-effective.

The training strategy has two objectives—the transfer of knowledge from the vendor's personnel and external consultants to the organization's key personnel and the dissemination of this knowledge throughout the organization. More precisely, the learning objectives establish what the learner should be able to do as a result of the training. Knowledge may be sought, particularly if related to operational best practices. One then has to ask how this knowledge can be used within the company. However, it is likely that the main thrust of training is upon the development of skill competencies in the use of the software functionality.

The executive committee needs to have sufficient understanding of what is involved in an implementation project so that it appreciates the potential problems

and can give the commitment and support that is required. This is particularly true for project sponsors. Their lack of appreciation of the issues may result in their viewing the implementation as just another project and lead to their distancing themselves from it. They may have a poor understanding of the roles and responsibilities of all the participants and when problems arise, may fail to appreciate that their involvement is required.

The members of the project team need to develop knowledge and skills that will enable them to establish how to best use the functionality for the operation and maintenance phase. Since the members of the project team will become the trainers of other employees, they need to develop the skill to be able to formulate and deliver a training course. Users need to have the skill for using the functionality relevant to their roles. They should understand the basic concepts of SCM and how to perform the day-to-day activities in the SCM system. Others who require training include managers, who should have at least an appreciation of what the system does. Ideally, project managers should have a good understanding of all aspects of the system so that they can be effective in dealing with any issues raised.

A select number of people will require more specific technical training so that they can design databases, write scripts, manage users, generate reports, and query the database for specific ad hoc requirements.

System administrators need to be able to set up the system and then maintain it. They will require knowledge about how to handle system security and deal with technical problems. They will need to develop a level of understanding of the functionality so that, at some stage after implementation when the project team is disbanded, they are able to manage the system smoothly.

Additionally, over time it can be expected that the SCM tool will evolve to some degree along with the company and projects that it serves. From time to time, it may be necessary to conduct additional training sessions to keep everyone abreast of the changes that have been implemented.

Success Factors

The biggest impact of SCM implementation is on the corporate culture, and the success of an SCM implementation depends on the people using it. To change the skills, habits, and attitude of the employees toward the SCM system and to make the employees see the potential benefits of the SCM system for themselves and the organization, the organization should give them proper training and assurance that their jobs and careers are safe and that they will benefit from the changeover. Training has to play a critical role in the success of an SCM system—during its implementation and during its operation. Six steps to handle the cultural changes that arise from implementing a new technology (like the SCM system) are described as follows:

1. *Start early:* Though the time frame in which employees need to change work habits may be short, you can prepare them for the new system and reduce the emotional impact of the change by starting early with targeted information campaigns and good training.

2. *Align the leadership team:* Make sure they not only understand the nature and benefits of the SCM system, but also the issues surrounding employee acceptance. The leadership should also constantly reassure employees and promise that they will do that is possible to make the transition to the new system as smooth and painless as possible.
3. *Set reasonable expectations:* Employees should understand that, in the short term, they are only expected to achieve a basic level of competency. Beyond this transition period, there will be many opportunities to excel and provide unique contributions. Setting high expectations will demoralize the employees when they fail to achieve them. It is always better to set realistic expectations, provide the right environment for the employees to excel, and then reward the excellence. Also, inform the employees that there is a settling down period for things to fall into the groove and advise them that only after the system has stabilized will they see the true benefits of the new system.
4. *Communicate specifically and continually:* Each level and area of the organization has its own needs and issues. Develop a plan that addresses the unique situation of specific groups but uses consistent and ongoing communication methods to keep staff updated on progress and involvement opportunities during the implementation period. Whether it is good news or bad news, convey that to the employees and inform them about the steps the organization is taking to deals with the problems, if there are any.
5. *Identify new work teams and roles early:* SCM systems bring large adjustments to work roles and responsibilities. Identify and communicate these changes early, so employees affected fully understand and mentally prepare for the new environment.
6. *Develop competency:* Competency will provide comfort to most employees facing the new environment. Make sure everyone has a solid overall understanding of how work flows through the new system. Then allow weeks of hands-on practice performing new job tasks in the system after formal training.

Three key elements to effective SCM implementation training for employees are described as follows:

- *Train the trainer or, in other words, SCM end users:* All organizations will have employees who have an aptitude for teaching and training. These people should be identified and trained to train others. These trainers will have more credibility and acceptance among the employees as they are part of the workforce. As these people have worked with the company, they have hands-on experience and are able to identify with the trainees and answer their questions better than the external consultants or trainers. In addition, these internal trainers can be leveraged to provide functional support during and after go-live.
- *Allocate plenty of time for SCM implementation training:* Many companies do the employee training as an afterthought once the implementation is over. This is a sure recipe for disaster. Effective training is arguably the most

important and critical part of the SCM implementation. Rushing the process and compromising on the quality or duration of the training program can result in costly failures.

- *Reinforce training with more comprehensive organizational change management activities:* Discussions surrounding employee changes should begin well before end-user training. As new business processes and changes are defined, employees should be kept in the loop so they're not blindsided during formal training—or worse yet, at go-live. Top management should address and resolve all issues like employee fears, uncertainties, and doubts about the new system well in advance so that the employees are fully behind the SCM implementation and participle in the training programs without any distractions.

SCM projects can go a long way toward making adoption easier when focused on effective training that ensures that employees are not overwhelmed, and deficient, at cutover. When properly prepared, most employees will support changes to the corporate culture, embrace the new SCM system, and adopt new values that ensure success.

Employees and Employee Resistance

Primarily, SCM implementation is not a technology or process project—it is a people project. To succeed during and after implementation, the SCM system needs full and complete support of all the end users of the system. The end users are the people who will be using the SCM system once it is in place. These are the people who were doing the functions that are being automated or computerized by the SCM system. With the implementation of the SCM system, the old job descriptions will change and the nature of the job will undergo drastic transformation. It is human nature to resist change. When we are talking about implementing an SCM system, we are talking about change on a very massive scale. So, there will be fear of the system replacing existing jobs, as many functions will be automated. Also, people will be afraid of the amount of training they have to undergo and learning they have to do to use the new system. Job profiles will change; job responsibilities will undergo drastic alterations; and people will be forced to develop new skill sets. If these fears are not addressed and alleviated well in advance, you are asking for trouble.

It should be worth noting the fact that while SCM systems eliminate many existing jobs, they create many new ones—with more responsibilities and value addition. It is easy to see that the automation of the business processes through technology can eliminate the jobs of many employees whose function is to record, control, calculate, analyze, file, or prepare reports. However, it must be pointed out to the employees that the same automation also creates many more opportunities for employees because they can get away from the monotonous clerical work and transform themselves into highly valued individuals in a new and challenging working environment using the most modern technology. If the company can succeed in making its employees accept this fact and assist in making the transformation (by

giving them training), then the major (and most critical) obstacle in the path of an SCM implementation is solved.

Reasons for Employee Resistance

Change is happening faster that most people care to think about. What is more important, change is happening faster that most people care to accept. In fact, most people do not want change! The premise is that change is not always good—that somehow it will have a negative impact. Today, technology is exponentially advancing. It is very difficult to keep pace and stay abreast with these technological developments. We are in a constant state of change, and continuous change and continuous improvements in the abilities of companies to do things better, faster, and cheaper is an absolute must for survival in this brutally competitive world.

The main reasons for the resistance toward change are fear of failure, fear of being redundant, and fear about the uncertain future. We will now examine these reasons in more detail.

Fear of Being Redundant

The biggest fear shared by people in companies going in for SCM implementation is the loss of their job. As soon as the decision about the SCM implementation is announced, rumors about the new system automating all the tasks and making people redundant will start floating around. When a company talks about SCM and automation, the immediate reaction is that computers will replace people. There is some truth in this fear. There are many instances where computers have made people redundant. However, what most people fail to hear and the gossipmongers forget to tell is that the people who were doing the manual jobs before computerization were able to get better jobs with higher salaries once they learned the new system and how to use the computers. So if a person is willing to adapt to the changes and to learn the new systems and new way of doing business, then he or she does not have anything to fear. There is a very good chance of getting better jobs with higher salaries.

Fear of Failure

Another fear that must be addressed in the planning phase is how to handle people's fear of failure—the fear of not understanding or being able to work within an automated environment. Many companies make the serious mistake of not insisting on a very thorough training program that will ensure the employees have the knowledge and a confidence level for adapting and using the new systems to the maximum benefit of the company.

Many view training as an expense to be cut. Beware of those that claim they can reduce this expense as the results have proven increases in hidden costs to the firm that often exceed the perceived savings. Training is an item that should receive the most serious attention by professional implementers. Once it is determined in

scope and budgeted for, it should become a sacred cow and be adjusted only by the implementer due to project scope changes.

Fear of the Future

Openly discussing and announcing the purpose of implementation and what it means to the employees of the firm can help to address the fear of the future. Normally SCM implementation investment is made so the firm can compete and grow the business. It may be implemented to survive, thrive, and become competitive.

Whatever the future expected from the implementation, it must be made openly clear to all who are expected to participate in making it successful. A feeling of excitement must be built from the ground up, in order for people to enthusiastically embrace the SCM system as the key to their future. Without a feeling of confidence that things are going to be good, they may never try it at all! If they decide not to give the system their full support, you will not be successful in your implementation.

Dealing with Employee Resistance

Implementing an SCM system is a change, and it is human nature to resist change. Accordingly, any SCM implementation will face some amount of resistance. The main reason for this resistance is fear—fear and uncertainty about what will happen. In the case of SCM implementations, too, there will be fear among the employees. There will fear about such things as what the new system is all about, what changes it will introduce, how it will change job profiles, how many jobs will be made redundant, and how many employees will lose their jobs.

It is quite natural that end users will be skeptical about the new system. However, for an SCM implementation to succeed, the cooperation of everyone involved is an absolute necessity. If employees are not convinced about the importance of the SCM system and the benefits of using it, they will not be fully cooperative. This can result in the failure of the system. It is very important, therefore, that users be won over before implementing the system. Forcing the system on unwilling people will only harden their resolve to revolt. The best way to deal with employee resistance is to educate employees about the new system and assure them about their jobs. The following sections examine some of the techniques used to deal with employee resistance.

Training and Education

One main reason for the resistance is ignorance. People always have a lot of misconceptions about SCM—e.g., it will increase work load, it will make jobs redundant, it will change the way business is done, or it will introduce limitations on the freedom of the employees. More problems have arisen during implementation from all levels of a company because people were uncertain of their future upon completion of a successful implementation. However, if the SCM implementation team backed by the management spends a little time and effort educating users

about SCM and how it will help both the company as well as the users, then user resistance can be reduced—if not fully eliminated. Remove the uncertainty for all in the company by telling personnel what is going to happen after the implementation of the SCM system.

Very few disagree that training is important. The problem, however is, that many projects tend to focus on training users on how to use the new system prior to go-live. While this is certainly important, it does not address other issues such as how key processes will be affected by SCM. In addition, not enough companies take advantage of ongoing training tools such as on-line help and ongoing refresher training.

Training is one of the most dangerous and expensive items to be compromised or circumvented. The lack of preparatory and reinforcement training or inadequate training can cause wrong and inefficient use of the system and result in very direct damaging expense to the firm and even more dangerous indirect expense.

Implement an Organizational Change Management Program

An organizational change management program is much more than just training. Organizations should ensure that they are holding regular workshops with end users and management, involving them in process and organizational design workshops, and keep users informed of such information as the purpose of the project, the progress, and why SCM is being implemented through formal and informal communications.

Creating SCM Champions

Another method of reducing resistance is by creating champions. According to Mosely [1], one of the most efficient ways to transition to new technology is to find a well-respected potential user of the technology. This should be a person who knows the business well, embraces change, and is respected in the organization. Train the user on the process and the technology, have this user evaluate the technology, and encourage this user to champion the merits of the technology to coworkers and management.

The champion becomes the expert user, facilitator, and trainer of the tool. He or she will also be the key to help other employees understand and learn the value of SCM and how it affects their jobs. Accordingly, all the members of the implementation team and the pilot project team are potential champions. There will always be people who adapt to change slowly and maybe even begrudgingly; do not look to them to be your champion. Instead, look for the people who are the first to embrace change and adopt new technology and who are always looking for a way to do things better. That is the kind of person you want for a champion.

Pilot Projects

Implementing the SCM system in a pilot project is a good idea because it minimizes the risk of failure. This is because the entire implementation team can concentrate on the pilot project. The SCM system can be tested before going in for company-wide

deployment. Any issues that were not anticipated during the planning stage that are encountered during the pilot implementation can be considered, and the implementation plan can be refined and fine-tuned. During the pilot implementation, the existing data of the pilot project is migrated to the SCM system; team members are given training on the SCM concepts and how to use the SCM system (and tools, if tools are used). The implementation team monitors the various implementation issues such as how people find the system, their feedback, tricky issues in the implementation, the learning period, how long it takes for the users to get comfortable with the system, and whether the user manuals and other implementation documentation are satisfactory or need revisions or modifications.

Based on the experiences of the pilot project implementation, the implementation plan and the implementation guide will be revised and modified. The pilot project will warn the implementation team about potential problems and how such pitfalls could be avoided. Also, a successful pilot project is a morale booster for the implementation team and a good marketing tool.

Involve Employees in SCM Process

The more involved employees are in the SCM decision and implementation, the more ownership and buy-in they will have into the project. This is not to say that every single employee should be involved. However, involving more employees than just senior management in the decision and implementation planning process will go a long way to make people feel more ownership, which inevitably results in less resistance in the future.

Address Issues of Fear, Uncertainty, and Self-Esteem

Each and every one of the levels within an organization must be addressed in overcoming resistance to change. The first thought in anyone's mind when SCM implementation is discussed is, "What's in it for me?" If the answer must be self-generated, then it will probably be on the pessimistic side, yielding a negative outcome perception. This critical issue should be addressed early in any project and at every level.

There is no such thing as communicating too much with employees. The more they know about why your organization is selecting SCM, how it will benefit the company, and what it means to them and their jobs, the less resistant they will be to changes when they are implemented.

Factual representations of what will happen to people within the organization, how the implementation will take place over what period, milestones during the project, and managing expectations of all (from the shop floor worker to senior manager) are absolutely necessary if a successful on-time and on-budget project is the goal.

Manage Expectations

Managing expectations of problems to be expected during the initial and final phases is the biggest challenge. Due to such factors as lack of top management commitment, spread of rumors, mistakes on the part of the implementation team,

and actions by the pessimists and skeptics, your implementation efforts can be derailed. Never let this type of activity impact a project. It will result in failed implementations if allowed to go unchallenged. So, the project manager and the implementation team should monitor the mood and attitude of the employees and their expectations and concerns and take corrective action before problems get out of control.

In the planning stage, these elements must be addressed and be well-documented for later firefighting, if necessary. If planning is thoroughly completed prior to implementation, most of these elements will be insignificant to nonexistent because they will have been dealt with early on. Further, if expectations are openly adjusted and well-documented during changes in project scope or circumstances, you can avoid surprises and be ready to deal with any problem that arises. If you are not prepared, then sudden and unexpected developments can surprise you, and by the time you formulate a strategy and implement it, it will be too late and usually can result in the loss of confidence in the project or you or both.

Contract with the Employees

The employees who are on the implementation team are trained on the SCM package at the company's expense. Once they have acquired the knowledge, completed the training, and participated in the implementation, their market value will improve exponentially. So it is natural that they will find better and more lucrative job offers. However, if these key employees leave the company without any warning or without making any alternative arrangements, then the company's performance will suffer. So, it is in the best interests of the company to sign a contract with the employees before they are put on the implementation team and given training. The main clause of the contract with the employee should be that they should not put the company in a position where the smooth running of the SCM system is interrupted. If they want to leave, they can leave; even the most stringent contract cannot hold them back; but before leaving they should give the company enough notice. They should also train another person to a level where he or she can handle the duties of the person leaving. The contract can also stipulate that no employee can leave the company in the middle of the implementation project, whether it is their first or fifth. The chances of an employee leaving during his or her first implementation are rare. It is once they have gained experience that they want to leave. Suppose the company has implemented the system in two divisions and is planning implementation in the third and suppose that the implementation is halfway. If, at this point, a key employee leaves, it will affect the implementation very badly. The contract with the employees should take such situations into consideration.

One word of caution, however, if the company can retain the employees by other means like offering attractive salaries, stock options, and a challenging and comfortable work environment, then adopting such a strategy is a much better way than enforcing a contract. Employees are the most important asset of any company, and it is in the interest of the company to trust them and keep them happy and satisfied.

Company-Wide Implementation

Once the pilot project has been successfully implemented and the implementation strategy and other items such as user manuals and technical guides have been revised and modified, the implementation team can proceed to company-wide implementation, in which the SCM system is implemented in all the projects in the company. This involves (1) training users in SCM and SCM tools and procedures, (2) migrating data to the tool repositories, (3) assigning roles and responsibilities to the project team members wherever necessary, and (4) monitoring the SCM system until it reaches a stable state.

As SCM is implemented in more and more projects, and as people learn of the benefits of an SCM system, the job of the implementation team will become easier. Project team members should be given adequate training, and there should be enough documentation (e.g., user manuals, FAQs, and how-to guides) so that new people joining the project will not have difficulty getting the necessary training and information regarding the SCM tool and SCM concepts.

SCM Implementation: The Hidden Costs

SCM implementation promises great benefits, but what are the costs involved? Exactly how much will a company have to pay to have an SCM system? In most cases, the SCM implementation costs exceed the budget. Why is this? Even a well-planned and thought-out budget is often exceeded. This section examines the areas that most planners miss accounting for in their budgets—in other words, we discuss the hidden costs of SCM implementation.

Although different companies find different hurdles and traps in the budgeting process, those who have implemented SCM systems agree that some costs are more commonly overlooked or underestimated than others. Armed with insights from across business, SCM implementation veterans agree that one or all of the following four areas are most likely to result in budget overruns: (1) training, (2) integration and testing, (3) data conversion/migration, and (4) external consultants. Each is discussed in the following sections.

Training

Training is the unanimous choice of experienced SCM implementers as the most elusive budget item. It is not so much that this cost is completely overlooked as it is consistently underestimated. Training expenses are high because workers almost invariably have to learn a new set of processes, not just a new software interface. Training is the first item that gets cut when budgets have to be squeezed—a major mistake, says most SCM implementers. A successful training experience will account for a minimum of 10–15% of the total project budget. Unwise companies that scrimp on training expenses pay the price later. Training costs cannot be avoided, but there are a few ways to keep the price tag under control. One way is train an

initial batch of employees who can then train their colleagues in turn. This solves two problems: (1) The huge training bills of consultants are reduced, and (2) because the training is done by their own colleagues, resistance to change is reduced and people will be more ready to accept the new system. In fact, it is a good idea to identify these would-be trainers early in the implementation phase and make them part of the implementation group, so that they will have hands-on experience and will understand the "big picture."

Integration and Testing

Today's SCM systems are very complex systems. Interfacing with those systems is not an easy task. Testing the links between SCM tools and other corporate software—links that have to be built on a case-by-case basis is another essential cost that is easily missed. Most companies will have some development environments that will not integrate with the SCM tool and will have to be separately interfaced. In most cases these integrations are costly.

Data Conversion or Migration

It costs money to move existing project information to the new system. Most data in most legacy systems is rubbish. However, most companies seem to deny that their data are dirty until they actually have to move it to the new client-server setups. As a result, those companies are more likely to underestimate the cost of data migration. Nevertheless, even clean data may demand some overhaul to match the process modification necessitated or inspired by SCM tool implementation.

Data Analysis

There is a misconception that SCM vendors spread: that you can do all the analysis you will want within their product. Often, however, the data from the SCM system must be combined with data from external systems for analysis purposes. Users with heavy analysis needs should include the cost of a data warehouse in the SCM budget and should expect to do quite a bit of work to make it run smoothly.

External Consultants

The extravagant cost of external consultants is a well-known fact. Like training expenses, this cost is hard to circumvent. Choosing a lesser known SCM tool to avoid premium-priced consultants does not necessarily help. When users fail to plan for disengagement from the existing system, consulting fees will overshoot the budget. To avoid this, companies should identify objectives for which its consulting partners must aim when training internal staff. It is a good practice to include performance metrics and time schedules for the consultants. For example, a specific number of the company's staff should be trained to a certain specified level of expertise within a specified time.

Brain Drain (Employee Turnover)

It is accepted wisdom that SCM success depends on staffing the project with the best and brightest from the business and IS. The software is too complex and the business changes too dramatically to trust the project to just anyone. The bad news is that a company must be prepared to replace many of those people when the project is over. Though the SCM market is not as intense as it once was, consulting firms and other companies that have lost their best people will be hounding yours with higher salaries and bonus offers than you can afford—or that your HR policies permit. Huddle with HR early on to develop a retention bonus program and to create new salary strata for SCM veterans. If you let them go, you will end up hiring them—or someone like them—back as consultants for twice what you paid them in salaries.

Continuing Maintenance

Most companies intend to treat their SCM implementations as they would any other software project. Once the software is installed, they figure, the team will be scuttled and everyone will go back to his or her day job. But after SCM, the implementation team members should not be sent back to their previous jobs as they are too valuable. They have worked intimately with the SCM system, SCM vendors, and external consultants. They know more about the sales process than the sales people and more about the manufacturing process than the manufacturing people.

Companies cannot afford to send their implementation project team members back into the business because there is so much to do after the SCM software is installed. Just writing reports to pull information out of the new SCM system will keep the project team busy for a year at least. And it is in this analysis and insight—that companies make their money back on an SCM implementation. Unfortunately, few IS departments plan for the frenzy of post-SCM installation activity and fewer still build it into their budgets when they start their SCM projects. Many are forced to beg for more money and staff immediately after the go-live date, long before the SCM project has demonstrated any benefit

Summary

In this chapter, we discuss how to implement the SCM system in an organization. We examine how to monitor the implementation project and why it is important to do the monitoring. We determine that one of the most critical factors in the success of an SCM implementation is the participation and cooperation of the users.

In addition, we discuss the different methods of making the implementation a success, including the role of external consultants, vendor participation, user education, and having in-house champions. We also describe the factors that result in cost overruns and how to tackle them. Furthermore, we consider the permanent nature of SCM systems and how the organizations should gear up to live with them and reap the full benefits from them.

The key message is that an SCM implementation is characterized by its complexity. There are lots of issues that need to be recognized and handled. Many can

be identified during the planning stage, but some issues will come up during the implementation. Accordingly, organizations should have a plan to deal with these uncertainties.

Every situation is unique. The complexity of the SCM implementation is such that one cannot anticipate the unexpected, particularly when people are involved. So the project management team should monitor what is happening and pay close attention to the details, no matter how trivial they might appear.

The SCM implementation cannot succeed without the cooperation of employees. Employees should be involved in every phase to ensure that they use the system properly and willingly. Most technical issues can be fixed. People problems are more difficult to fix, if they can be fixed at all. The evidence of people problems are employee turnover, disputes, and low levels of motivation. Accordingly, the implementation should place the highest priority on employees and their problems.

Reference

[1] Mosley, V., et al., "Software Configuration Management Tools: Getting Bigger, Better, and Bolder," *Crosstalk: The Journal of Defense Software Engineering*, Vol. 9 No. 1, Jan. 1996, pp. 6–10.

Selected Bibliography

Bourque, P., and R. E. Fairley, eds., *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

Buckley, F. J., "Implementing a Software Configuration Management Environment," *IEEE Computer*, July 1994, pp. 56–61.

CM Crossroads: The Configuration Management Community (http://www.cmcrossroads.com/).

Dart, S., "Achieving the Best Possible Configuration Management Solution," *Crosstalk: The Journal of Defense Software Engineering*, September 1996.

Dart, S., "To Change or Not to Change," *Application Development Trends*, Vo1.4, No. 6, 1997, pp. 55–57.

Dart, S., *Configuration Management: The Missing Link in Web Engineering*, Norwood, MA: Artech House, 2000.

Estublier, J., "Software Configuration Management: A Roadmap," *Proceedings of the Conference on the Future of Software Engineering*, Limerick, Ireland, 2000, pp.279–289.

Feiler, P. H., "Software Configuration Management: Advances in Software Development Environments," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1990.

Irish, D. E., "Putting the Horse Before the Cart: Preparing Your Staff for Project Management Software," *Proc. ACM SIGUCCS 2001*, Association of Computing Machinery, 2001, pp. 59–62.

Kolvik, S., "Introducing Configuration Management in an Organization," *Proc. ICSE '96 SCM-6 Workshops (Selected Papers)*, Berlin: Springer-Verlag, 1996, pp. 220–230.

Moor, S. R., J. Gunne-Braden, and K. J. Gleen, "Enterprise Configuration Management—Controlling Integration Complexity," *BT Technology Journal*, Vol. 15, No. 3, July 1997, pp. 61–72.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

Tellioğlu, H., and I. Wagner, "Negotiating Boundaries: Configuration Management in Software Development Teams," *Computer Supported Cooperative Work (CSCW): The Journal of Collaborative Computing*, Vol. 6, No. 4, 1997, pp. 251–274.

Thompson, S. M., "Configuration Management—Keeping it all Together," *BT Technology Journal*, Vol. 15, No. 3, July 1997, pp. 48–60.

CHAPTER 18

The Different Phases of SCM Implementation

Introduction

We have learned that SCM is a set of activities that must be performed throughout the life cycle of the software system. Even though SCM activities can be initiated at any stage during a project's life cycle, it is better to have the SCM system in place from the very beginning.

Company management should be convinced about the need for and importance of having an SCM system, preferably during the early stages of the project. Also, the people who will be using the system—including the developers, the project leaders, the QA team members, and company management—should be made aware of the benefits of a good CM system. The benefits of SCM as discussed in Chapter 4 could be used for this purpose.

SCM systems are capable of delivering dramatic productivity improvements, cost reductions, and error or defect reductions, among other benefits. SCM can improve customer goodwill, because the company will be able to provide customers with better quality products and better technical support. By educating project personnel on the benefits and values of SCM, the myths about SCM—such as, "SCM is nothing more than just additional documentation," or "SCM is additional work"—are dispelled and the need for SCM becomes obvious. Also, today's SCM tools are so sophisticated and advanced that they make the whole process of performing SCM functions much easier than before.

There is a misconception that SCM is only for big companies and large projects. In the author's opinion, SCM should be implemented in all software projects—irrespective of the size of the project and organization—for the very reason that change is inevitable in all projects, and unmanaged and uncontrolled change is trouble all the way. Companies that use the scientific methods of software development from the very beginning—that is, even when they are small—have a workforce and work culture that is more willing and able to learn new technologies and adapt to changes and implement new procedures. The programmers and managers of these companies have gotten used to standard software engineering practices and procedures, and the work cultures of these companies evolve around these practices. New employees joining the company will also follow the procedures and methods, because peer pressure will be high.

The advantage of this scenario is that the methodologies and procedures will produce dramatic results in such companies, because people are doing things because

they believe in them, not because they have to do it. Consider SCM; everyone knows that SCM is not easy. It is difficult to keep a good SCM system (a system that is efficient and effective) up and running. However, the effort is worth it. Still, if an SCM system is to become a success and deliver the promised results, the people who are involved have to be convinced of its worth. Just doing SCM for the sake of getting some certification will not produce the results that it is capable of.

This chapter aims to explain the objectives and phases in the life cycle of an SCM system, from conception to retirement, and discuss how these events may be sequenced. The organizational culture and the nature of projects will differ from company to company. So two SCM implementations can never be identical; they will vary depending on factors such as the size, nature of projects, complexity of projects, development methodology, and organizational culture. The objectives, the different phases, the sequence of these phases, and other details of the SCM implementation will vary greatly from organization to organization.

It is important for company management, project leaders, consultants, developers, SCM personnel, and other stakeholders to have a good understanding of the large-scale conceptual picture of SCM and SCM implementation. Without the capability to understand the big picture, SCM practitioners can become lost. It is important to have a solid foundation for understanding the big picture so that the practitioners can understand the details of what they deal with on a day-to-day basis and how it relates to the rest of the project and impacts on the rest of the organization.

There are two basic characteristics common to all SCM implementation projects—objectives and phases (or events).

Objectives of SCM Implementation

Objectives are the major high-level characteristics that can have a great impact upon the success of an SCM project. The objectives include characteristics such as the following:

- Speed;
- Scope;
- Resources;
- Risk;
- Complexity;
- Benefits;
- Speed.

Speed of a project is directly related to the amount of time that a company has before the completion of the SCM implementation or the amount of time that the company would like to take for the implementation. The speed of the project in the context of this chapter is how much time the company would like to take in implementing the system. The amount of time that the company actually takes may be dramatically different. The amount of time that the company would like to take should be the figure used when developing a project plan.

Scope

The scope of the project includes all of the functional and technical characteristics that the company wants to implement. A company installing a full-fledged SCM system would have a much greater scope than a company installing a change management and defect tracking system.

Resources

Resources are everything that is needed to support the project. This includes people, hardware systems, software systems, technical support and consultants. All the different resources of an implementation have one thing in common—money.

Risk

The risk of a project is a factor that impacts the overall success of the SCM implementation. Success is measured by factors such as overall user acceptance, return on investment (ROI), time to implement, etc. High-risk situations are less likely to possess these characteristics.

Complexity

Complexity is the degree of difficulty of implementing, operating and maintaining the SCM system. Companies of different sizes, business environments and organizational cultures have different levels of complexity. A multi-billion dollar corporation that has development sites and teams spread across the different parts of the world and working in different time zones is generally much more complex than a company with 50 employees occupying one geographical location.

Benefits

Benefits are the amount to which the company will utilize functionality of the SCM system for software development, maintenance and other support activities. The SCM tools automate almost all aspects of the CM activities; they make the job of the developers, managers, and other stakeholders easy and improve the development productivity. To get the maximum benefit out of an SCM implementation, the system should be built around the software development process and organizational procedures followed by the organization. Better integration of the SCM system with the software development and maintenance process will result in high-quality products, reduction in number of defects, faster bug fixes, quicker incorporation of enhancements, better customer service, etc. which leads to improved customer satisfaction and goodwill. This will result in satisfied customers and improved brand image, which will lead to an increase in market share and profits.

Each of these objectives can be rated on a scale from low to high. Interrelationships exist between the different objectives. For example, companies that attempt to install an SCM system with low resources and high complexity and at high speed would place themselves at high risk. Readjusting risk to a low value would cause

other objectives to change their value based on other factors such as complexity. In addition to the interrelationships, there exist various dependencies between the objectives. A good example of such a dependency is between speed and risk of the SCM implementation. A quickly implemented SCM system tends to be at higher risk than one implemented at a slower pace, taking the necessary precautions. When deciding on an SCM implementation plan, the various objectives and their interrelationships and dependencies should be taken into consideration.

Different Phases of SCM Implementation

Like any project, SCM implementation goes through different phases. There are no clear separating lines between these phases, and in many cases, one phase will start before the previous one is completed. Still, however, a logical order is followed. The logical order is the order in which the phases are listed. As mentioned, some overlap occurs between the phases, but the third phase cannot start before the second phase and so on. There are two way to implement an SCM system in an organization—company-wide implementation (also known as the "big-bang" approach) and project-by-project implementation (the incremental approach). Irrespective of whether an organization chooses the big-bang approach or incremental approach, the SCM implementation phases described in the next sections will have to be done for each project, as each project has its own peculiarities and needs. The advantage of the big-bang approach over the incremental approach is that many phases, such as training, tool selection, and tool implementation can be done for all the projects simultaneously, which can result in the reduction of training and implementation expenses.

Here, do not confuse SCM implementation with the installation of the SCM tool. The SCM tool used for each project can be the same but need not be. Usually, when an organization chooses an enterprise-wide CM (ECM) tool, all the projects in the organization will be using the same tool, but that may not always be the case. In such cases, the tool implementation will be done only once and as and when new projects are started, depending on the specific requirements of the project, factors such as the controlled libraries and workspaces are allocated.

One thing that should be remembered is that the SCM system is unique for each project, and the phases described in this section will have to be repeated for each project (maybe with the exception of tool installation). Perhaps, as time goes on and as most of the employees are trained in SCM basics and SCM tools, the time and effort spent on training might decrease. However, we have to consider that new employees who join the organization will have to be trained. Furthermore, in a given project, new team members will need to be trained. Also, there will be changes such as product upgrades, new functionality in the new versions of the tools, new policies, and new standards and regulations. All of these factors will require further employee training. Accordingly, as more and more employees get trained in SCM, the cost of training will come down, but the training activities will have to continue—training is a never-ending process. SCM tools will last through several years and many projects and will be retired only when they become obsolete. This is discussed further in the section on SCM tool retirement.

It is also important to remember that each of the phases discussed in the following sections consists of many subphases or events. These events will be discussed in later chapters. Some of these events are needs (requirements) analysis, request for information (RFI) and general research, ROI analysis, request for proposal (RFP), reference site examinations, hardware sizing, vendor site surveys, SCM tool selection, contract negotiations, customization decisions, documentation, end-user training, audits, performance measurements, and ongoing education and maintenance. Note that all of the phases discussed in this chapter may not be applicable in all cases. For example, in a small project where a single person is responsible for SCM, there will not be an SCM team and SCM team training; or if the organization has already identified an SCM tool, then the preselection screening and tool evaluation phases (part of SCM system design) are not required.

The different phases of the SCM implementation are listed as follows and illustrated in Figure 18.1:

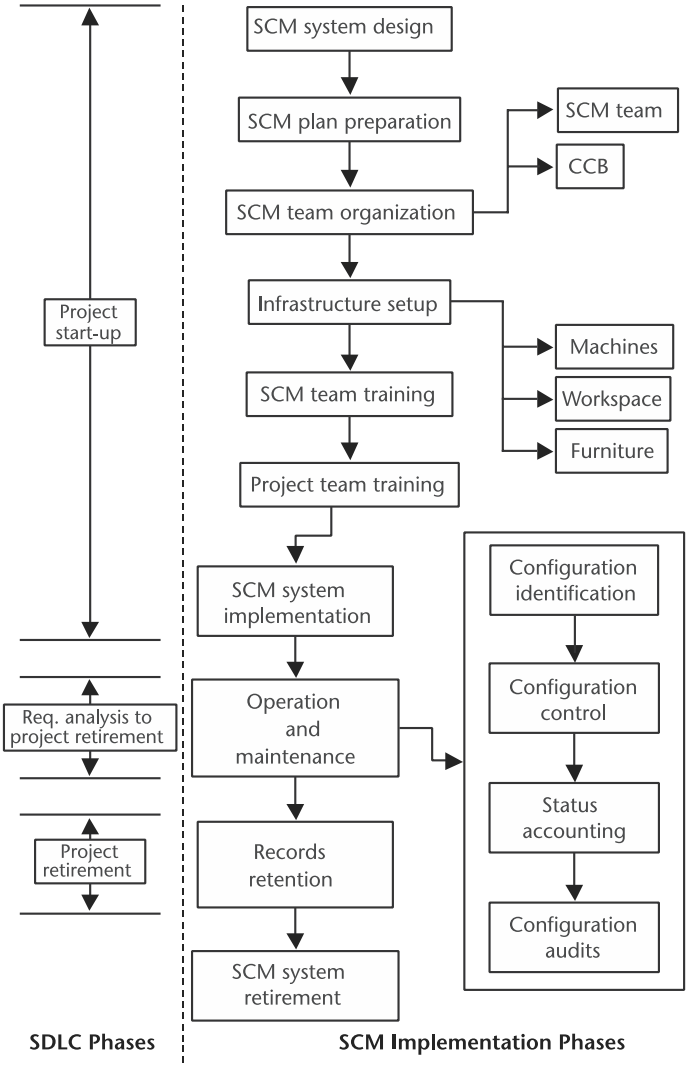


Figure 18.1 Different phases of SCM implementation.

- SCM system design;
- SCMP preparation;
- SCM team organization;
- SCM infrastructure setup;
- SCM team training;
- Project team training;
- SCM system implementation;
- Operation and maintenance of the SCM system (configuration identification, configuration control, CSA, and CAs);
- Records retention;
- SCM system retirement.

The initial phases—from SCM system design to SCM infrastructure setup—are performed by the SCM system design team with the help and support of company management. SCM team training is done jointly by the SCM design team and SCM experts (outside consultants or in-house experts). If SCM tools are used, then the training on those tools is given by the tool vendor's representatives or in-house tool experts.

The project team members are trained by the SCM team members, the design team members, the tool vendor's representatives, and SCM experts. Here the role of the SCM experts and the SCM system design team members is to develop a loyalty among employees for SCM. Because these people will be quite senior in the organizational hierarchy and have the necessary stature, they can convince people better than the SCM team members. So these top-level people should give their support, pledge their allegiance, and give an overview of SCM and its benefits to the employees. Training on the details and the day-to-day operational formalities can be handled by the SCM team members. The support and encouragement of top management and the full cooperation of all people involved are essential factors in the successful implementation and smooth functioning of any SCM system.

A question that naturally arises is: how can SCM team members and project team members be trained on the SCM system before the system is implemented—,that is, how can somebody be trained on some thing that does not exist? Actually the system exists, but it has yet to be implemented. The SCM system is designed; procedures are documented and defined in the SCMP; and the tools that are going to be used are selected, purchased, and put in place. The SCM team and project team training takes place so that people can use the systems properly. Accordingly, the main objective of the training is to establish the best practices and to convince users about the need for, importance of, and benefits of the SCM system—the one they are going to use.

The SCM system implementation phase involves installing the SCM tools and assigning duties and responsibilities. Once the tools and the necessary procedures are in place, the SCM team, developers, QA personnel, and other parties who will be using the system will work under the supervision of the consultants and tool vendor representatives to iron out any problems that might arise and to fine-tune the system. Once the problems are identified and solved, the system starts working smoothly, and the employees become confident about running the system, it is

handed over to the employees. From this point onward, the operation and maintenance phase starts; it will last until the system is retired.

Although the SCM implementation phases may seem very linear and distinct from each other, in reality, throughout an actual implementation, the phases are quite fluid. In addition, the phases are repeated. For example, even the SCMP can undergo changes. So if it is felt that the current system needs changes, the changes are incorporated into the SCMP, and if that change necessitates training, then that is done. Everyone involved in the SCM process is informed about the changes made to the SCMP. The change management procedures are repeated many times—every time a change request is initiated. Status accounting, the function that records the happenings and reports the information, is a routine task. SCM audits and reviews are also done quite often depending on the criteria specified in the SCMP.

We now examine each phase in some detail.

SCM System Design

Once project management or company management decides to support SCM, the SCM system must be designed. If the company already practices SCM and already has guidelines, then the job is easy. The job of the designers is to tailor the company guidelines to suit the needs of the particular project.

Here, one important thing to remember is that no two projects are the same. So even if the guidelines are taken from similar projects, it is necessary to customize them to suit the needs of the current project. The standards and guidelines must be customized depending on the nature of the project—some portions might need to be modified; some project-specific things might need to be added; and unwanted portions might need to be deleted. For example, if the guidelines talk about subcontractor control, and the current project does not have any subcontracted items, then that can be deleted.

The SCM system design team should include the project manager, the person who is going to be the SCM team leader, and key personnel from the project team and the QA department. It is not a good idea to have too many people on the design team; it will only result in lowering the productivity of the team. The responsibilities of the SCM design team include (but are not limited to) development of the SCM system, customization of the SCM system (if guidelines already exists), preparation of the SCMP, maintenance of the SCMP, selection of the SCM team, and constitution of the CCB.

If the company does not have any guidelines for the development of an SCM system, then SCM standards can be used, as discussed in Chapter 11. The design team defines the scope of the SCM system, what activities it will perform, how they will be performed, which activities will be automated, what tools will be used, what method will be used for version numbering, and how release management will be accomplished, among other issues.

The SCM system design team also makes decisions such as whether to use tools and whether to make them or buy them. In addition, the SCM system design team determines whether to use a manual system or an SCM tool. The decision about whether to make or buy the tool is also taken by the design team. If the team decides

to buy tools, then it will evaluate the tools available in the market and select the tool that is best suited to the needs of the company.

The team also decides the composition of the SCM team. The team size varies depending on the nature and size of the project. Large projects will have a full-fledged team with many people, whereas a small project may have a team comprised of a single person or a person working part-time. The design team also determines the constitution of the CCB and defines guidelines for its functioning.

Once all issues regarding the SCM system have been finalized, the details are documented. This document is called the SCMP.

SCMP Preparation

As we have seen, the SCM system design team decides on the particulars of the SCM system that is to be used. Once the system details are finalized, the decisions and procedures have to be documented. This document is called the SCMP. The idea behind the SCMP is to ensure that all members of the SCM team and the project team are aware of the procedures and the duties and responsibilities that each one is supposed to carry out. It will also tell them what resources are to be used and how they are to be used. It acts as a guideline in the resolution of conflicts. According to the IEEE [1], the SCMP should contain the following sections: introduction, management, activities, schedules, resources, and plan maintenance. Chapter 1 introduces SCMPs, and they are described in more detail in Chapter 13.

The SCM design team prepares the plan and distributes it among the members. This document forms the basis for SCM training. The plan also contains a section that lists the procedures required to keep it up-to-date. So the plan is constantly reviewed, and any required changes are made to it. The SCMP is also an item that is placed under configuration control, which happens once the plan is finalized, reviewed, and approved. It usually forms part of the functional baseline, because the SCMP is typically created during or before the requirements phase. Accordingly, once a baseline is established for the SCMP, all changes to the plan will have to be made in accordance with the change management procedures.

SCM Team Organization

As we have seen, the SCM team size can vary from a single person to a full-fledged team depending on the many variables that can have an impact and influence on of the project. The SCM system design team selects the members of the SCM team and allocates responsibilities to each member.

The constitution of the CCB and its workings are also finalized. The CCB usually consists of the SCM team leader and one or two key team members in addition to the project team leader, a QA representative, a marketing team representative, and, in some cases, client representatives.

SCM Infrastructure Setup

While the SCM team is formed and their responsibilities assigned, the infrastructure facilities that will help the team function properly must also be arranged. The

SCM is not a one-week or one-month affair. It is a continuous function that will be there for the entire life cycle of the project. So the SCM team needs permanent facilities, not makeshift arrangements. The final form will ultimately depend on the size of the project and SCM team. Ideally, the SCM team should have a separate office that is close to the project with which it is associated. This type of a setup is required for manual SCM systems in fairly large projects.

Today, most companies use SCM tools, and more and more SCM functions are being automated. SCM tools give a lot of power and capabilities to the development team. However, it is important to remember that SCM tools have become more complex and sophisticated. Their capabilities are no longer limited to change management, defect tracking, or source code control. They perform additional tasks like automatic status journaling and status accounting, branching and interactive merging, and automatic check-out and check-in. The additional functionality and capabilities of the new-generation SCM tools have resulted in the need for highly specialized personnel—e.g., SCM administrators, database administrators, and build and release managers—to operate and maintain SCM systems. So in today's scenario, the SCM team consists of highly specialized personnel who devote their full time to managing and maintaining the SCM system.

SCM Team Training

The members of the SCM team may be veterans with many SCM projects under their belts, or they may be novices with no knowledge of SCM. The idea behind SCM team training is to familiarize the team members with the discipline of SCM and train to practice it in a particular project.

Team members are trained on how to carry out their duties and responsibilities in the most efficient and effective manner and told what they are supposed to do and not supposed to do. If SCM tools are being used in the project, then the team members trained on the tools also. In addition, they are briefed about their access privileges and rights. Because the CM process is a job that involves a lot of tact and diplomacy, such training should also have a module on effective communication.

As mentioned in the previous section, the increasing popularity of SCM tools and the level of automation that is being achieved by these tools have reduced the role of the SCM team. In a manual SCM system, where all SCM functions were carried out manually, full-fledged SCM teams were required. Today, however, with the high degree of automation and more and increasing number of capabilities and responsibilities being given to development team members, the number of tasks that need to be performed by SCM team members has declined considerably.

Project Team Training

The success of an SCM system depends on the participation and cooperation of the project team members and on the understanding and dedication of the SCM team. Accordingly, training project team members about the fundamentals of SCM and its concepts, advantages, and benefits is necessary. Also, project team members should be briefed on how they are to participate in SCM functions and carry out SCM activities. If the project uses automated tools for change management and

problem reporting and tracking, for example, then the project team members should be trained to use those tools.

Both the SCM team training and the project team training are continuous activities, and provisions should be made to keep it that way. Training never ends; it is an ongoing process. This is because new people will join the teams and existing members will leave. Accordingly, new members need to be trained, possibly by an outgoing member or some other person designated by the project management.

The training of both of these teams is based on the foundation detailed in the SCMP. The SCMP details how SCM is to be practiced for the particular project, and all training activities should be based on that.

SCM System Implementation

This phase involves the installation of the SCM tools and assignment of duties and responsibilities for the tool administrators and users. Once the tools and necessary procedures are in place, the SCM team, developers, and QA personnel and other parties who will be using the system will use the system under the supervision of the consultants and tool vendor representatives. These external consultants and vendor representatives will help the employees—e.g., SCM administrators, SCM team members, developers, QA personnel, testers, build and release managers, and management—to properly use the installed SCM tool. The SCM administrator needs to set up the databases, workspaces, and libraries and should configure the tool for the organization or project. The vendor representatives will guide the administrator in this process and help him or her to understand the intricacies of the tool. They will also give him or her training on how to scale the different parameters as the project and team size increases. The consultants and vendor representatives will also help customize the tool to suit the organization's development processes and procedures. Once management and employees become confident about independently operating and maintaining the SCM system, the services of the consultants and the onsite presence of the vendor representatives are terminated, and the system is handed over to employees.

Operation and Maintenance of the SCM System

After the external consultants and the tool vendor representatives leave, handing over the system to the in-house personnel, the responsibility for managing and maintaining the system will fall on all the users of the system. During this phase the major SCM activities—configuration identification, configuration control, status accounting, and CAs—are performed. These four activities or phases, which form the core of the SCM activities, are discussed in detail in Chapters 7–10.

The SCM tool administrator will be responsible for such tasks as the regular upkeep and trouble-free operation of the system allocation of disk space, and management of the controlled libraries. The SCM system administrator will also be in constant touch with the tool vendor, so that any new upgrades or patches that are released can be installed without delay. The SCM team members, if any, will be responsible for managing the day-to-day SCM activities like change request

processing, arranging reviews and audits, check-in and check-out, and build and release management. The developers, testers, and other support personal will follow the guidelines provided for carrying out the different operations like initiating change requests, making changes, performing impact analysis, and conducting audits. The project managers will use the system's querying and reporting capabilities for project monitoring and tracking. It is during this phase that the SCM system will produce dramatic improvements in development productivity and product quality and also in the organization's capability to react to change.

Records Retention

Before retiring the project or software system, the documents or records of the SCM system must be archived, retained, or destroyed. SCM systems accumulate a lot of documentation and records during the life time of the project. With the passage of time, the information contained in the documents declines in value. Such records are removed from active accessibility. Depending on the nature of the record, it is destroyed immediately upon deactivation or is kept in retention for a defined period of time. The documents that do not have any value once the project is retired are disposed of. The documents that have some value either due to legal or contractual obligations are kept in retention for the period specified by the law or contract. The documents that have a sustaining utility exceeding storage costs are preserved permanently in an archive.

SCM System Retirement

The final phase in the life cycle of an SCM system is retirement. During the retirement phase of the project, the SCM system for that project is also retired. Once record retention and archival work is completed, the SCM team members assigned to the project are released for other projects.

SCM Tool Retirement

Modern SCM systems use SCM tools for performing SCM activities. SCM tools support many projects and last several years. During this period, tools are upgraded as and when tool vendors release new versions. After many years of service, a stage is reached when any further maintenance would not be cost-effective. This could happen because the size, nature, and complexity of the projects have increased, or because new tools that offer considerably more advantages (e.g., a higher level of automation, smaller SCM teams, or increased functionality) have come onto the market, making the current tool obsolete.

Another reason for retiring an SCM tool is that technological advancements have made the existing tool obsolete. The hardware on which the tool runs has to be replaced by a different (more powerful and less expensive) machine with a different operating system, and it is cheaper to install a new tool rather than upgrading or modifying the existing one. In each of these instances, the existing tool is retired, and a new one is installed in its place.

Why Many SCM Implementations Fail

Many SCM implementations fail miserably during the initial stages of the operational phase itself or fail to deliver the promised benefits. Why does this happen? The following are some of the most common reasons:

- Lack of top management buy-in, commitment, and support;
- Improper planning and budgeting;
- Use of the wrong SCM tool;
- Lack of training;
- Work culture of the organization.

Lack of Top Management Buy-in, Commitment, and Support

One of the most common reasons for a failed implementation is lack of top management support. Top management must be clearly convinced of the importance of SCM, how it can be used as a competitive weapon, and how the company can fail if a scientific mechanism like SCM is not available to manage and control change. If management is aware of the potential benefits of SCM and dangers of not having an SCM system, it will give its full backing and the necessary organizational resources to implement the best SCM system possible. When employees know that the SCM implementation has full management backing, they will also want the system to succeed. There will be a lot of issues that arise when the SCM system is implemented, including change of procedures and reassignment of employees. If management can assure employees that their jobs are secure, that assurance will go a long way in ensuring employee cooperation. Top managers should also talk to employees regarding the benefits of the SCM system and how the company can get ahead of the competition by reaping the benefits of the system.

Improper Planning and Budgeting

Before starting the SCM implementation project, detailed planning involving all the major stakeholders is necessary for the success of the project. It is during this phase that the company needs to make decision such as which procedures to follow, which tools to buy, and how large a budget to allocate for implementation and maintenance. If this planning is not done properly, then many factors may be overlooked, resulting in such problems as selection of the wrong tool, insufficient funds, or inadequate team members. All these can lead to the failure of the project. To avoid this, companies should take the planning phase seriously and do meticulous research before taking any action. Also, there should be a provision in the plan to revise and update it as the implementation progresses.

Use of the Wrong SCM Tool

We have seen that no two organizations are the same and that each organization requires an SCM tool that is best suited for its organizational environment, work culture, and development procedures. Accordingly, the SCM planning team should

take all these factors into account before deciding on a tool. They should research the available tools, match them with the organization's requirements, visit companies where the tools are installed to see them in action, and discuss end-user training, tool updates, and upgrades. Only when all the members of the team are convinced that a specific tool is best suited for the organization should the team make the purchasing decision. *Never rush the purchasing decision; the time spent on researching and analyzing before the purchase is worth the effort.*

Lack of Training

One of the main reasons the SCM fails is due to the resistance of users. The resistance is often the result of ignorance and fear—ignorance about the tool and fear of additional work or unemployment. These factors can be corrected by giving proper training. Training should be given at different levels on different aspects of SCM implementation. The top management should address the employees' fear of losing their jobs as many tasks get automated by the tool. They should also pledge their allegiance to the SCM system and make it clear to employees that the SCM system is essential for the success of the organization in this highly competitive business environment. The SCM team members, tool vendors, and external consultants should explain the principles of CM and its advantages on a day-to-day basis, including how it reduces rework and defects. Most users think of SCM as a system that creates more paperwork or documentation. Such myths about SCM should be debunked. SCM tool vendors and external consultants along with in-house experts should train users in how to efficiently use the tool and explain how SCM tools can make their lives easier and help them create high-quality products without chaos and confusion. Once users are convinced of the potential of the SCM system, the system will succeed; without user buy-in, even the best SCM system will fail.

Work Culture of the Organization

The work culture of the organization is very important for the success of SCM. If the organization has a workforce that is willing to learn new things and change to new technologies, then there will be no problems for SCM implementation. However, if employees resist change and see the introduction of formal methods as a means to assign accountability, they will perceive the new technology as something negative. Accordingly, the basic mindset of the workforce needs to be changed. This is important not only for the success of SCM, but also for the success of process improvement initiatives like CMM, SPICE, and Bootstrap. To change the employee mindset, the two critical factors required are top management support and proper training.

Summary

SCM implementation changes the way people have been doing things, and lots of new procedures are introduced for the functioning of SCM. Resistance to SCM implementation is natural, because it is human nature to resist change. Making

people accept SCM and implementing SCM are difficult because of the myths surrounding SCM, such as ideas that it causes additional work and more documentation.

Most people are not aware of the potential benefits of SCM. For an SCM system to succeed and deliver those benefits, an organization has to design a good system, install procedures, and train the SCM team and project team. Once these tasks are done, there is a natural tendency to feel satisfied or complacent about what has been achieved by the implementation team.

This is why the postimplementation phase is very critical. SCM functions are continuous and should be performed throughout the life cycle of the project. To reap the full benefits of an SCM system, the system should get project-wide and company-wide acceptance. To get project-wide acceptance for an SCM system, every member of the project should be made aware of the need, importance, and benefits of SCM.

Just as courtships and honeymoons are different from marriages, living with an SCM system is different from installing it. Implementing a good SCM system is not an easy job, but it is how the projects mesh with the SCM system that determines the value that is received from it. It is how the SCM system is used in the project that makes the difference. Even a well-designed system can be a failure if the people using it are not cooperative.

Reference

[1] IEEE Standard for Software Configuration Management Plans (IEEE Std-828–1990), *IEEE Software Engineering Standards Collection 2003 (CD-ROM Edition)*, Piscataway, NJ: IEEE, 2003.

Selected Bibliography

Fredrick, C. R., "Project Implementation of Software Configuration Management," *Proceedings of the 1981 ACM Workshop/Symposium on Measurement and Evaluation of Software Quality*, 1981, pp. 49–56.

Wingerd, L., and C. Seiwald, "High-Level Best Practices in Software Configuration Management," Technical Report, Perforce Software (http://www.perforce.com/perforce/best-practices.html).

CHAPTER 19

SCM Deployment Models and Transition Strategies

Introduction

The different SCM deployment options available to an organization include traditional license or on-premises deployment, hosted deployment, software as a service (SaaS), or the on-demand model. The two transition strategies available are the big-bang and incremental strategies. This chapter discusses these different options and their advantages and disadvantages.

Traditional License or On-Premises Deployment

On-premises technology has the same meaning as owning your computer hardware and software. You own the equipment (hardware), and you own your software. Most of the software we have used in the past is on-premises and owned by us. It is just like buying an office building. You pay for your software once or finance it. Then you pay for the ongoing maintenance and upkeep. In this arrangement, you are responsible for maintaining it, for fixing it when it breaks, for making sure it is meeting your needs, and for keeping it up and running. Some organizations have internal IT departments to handle all of this, and others use outside companies to help them. Also, in many cases, there are multiple helpers (people or organizations) either internal or external.

Advantages of On-Premises SCM System

The advantages are listed as follows.

- They give the user the greatest control and flexibility, just like owning your own office building.
- You do not have to check with anyone if you want to make modifications.
- You control your data and manage it any way you wish.
- You can build your systems to your own specifications to meet the specific needs of your organization.
- A copy of the software runs on your computer equipment for your use and is not shared with anyone outside of your organization.

- You can leverage the existing investments and reduce costs by using the hardware and software you already own.
- With an on-premises solution, you can easily connect with legacy systems. While some cloud solutions can connect with existing equipment, on-premises solutions can always do so.
- On-premises solutions are not subject to fluctuations in performance or availability due to the Internet (although on-premises solutions are not necessarily faster than hosted ones) and can guarantee predictable performance.
- On-premises solutions can be faster than a remote-hosted solution due to the inherent delays in data traveling long distances. For any computing solution, latencies are due to two things—computational time and transmission time. An on-premises solution reduces the distances traveled to meters or less. A remote solution can be hundreds of kilometers away.
- You can ensure compliance with security and other policies that require on-premises solutions.
- You have complete control of who accesses your systems, your software, and your data.

Disadvantages of On-premises SCM System

The disadvantages are listed as follows.

- High initial costs;
- Long-term capital investment;
- Costly and complex software implementation, maintenance, and upgrades.

Cloud Computing

Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility over a network (typically the Internet) [1]. Having the ability to provision a server in minutes with the required applications preinstalled is just one of the ways in which cloud differs from the traditional way of ordering a server, where you had to wait for days to have the server operating system and applications installed [2].

The ability to dynamically provision and deprovision based on demand, going well beyond the ability to just quickly add or remove environments, makes the cloud the most advanced platform to dynamically respond to increases in demand and automatic allocation of additional capacity to meet the user load. So instead of providing physical hardware, a data center, and applications, we can do those things in a virtual environment. Virtualization is the creation of a virtual, rather than physical, version of something, such as an operating system, a server, a storage device, or network resources. Virtualization has made it possible for IT organizations to decouple logical infrastructure from physical infrastructure and thereby deliver more flexibility to provide and manage value-added services.

The National Institute of Standards and Technology's [3] definition of cloud computing identifies "five essential characteristics." They are described as follows.

- *On-demand self-service*. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without human interaction with each service provider.
- *Broad network access*. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
- *Resource pooling*. The provider's computing resources are pooled to serve multiple consumers using a multitenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
- *Rapid elasticity*. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear unlimited and can be appropriated in any quantity at any time.
- *Measured service*. cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Cloud Computing Models

Cloud computing is a general term for every computing model that involves delivering hosted services over the Internet. These services are broadly divided into three categories: infrastructure as a service (IaaS), platform as a service (PaaS) and SaaS. cloud computing providers offer their services according to the above mentioned fundamental models.

IaaS

IaaS is a provision model in which an organization outsources the equipment used to support operations, including storage, hardware, servers, and networking components. The IaaS provider owns the equipment and is responsible for housing, running, and maintaining it. The consumer of IaaS services usually has control over the configuration aspects of the resource, such as which operating system to run on a virtual machine or how to utilize the storage resource. The consumer typically pays on a per-use basis. To deploy their applications, cloud users install operating-system images and their application software on the cloud infrastructure. In this model, the cloud user patches and maintains the operating systems and the application software. cloud providers typically bill IaaS services on a utility computing basis; the cost reflects the amount of resources allocated and consumed.

PaaS

PaaS is a service that offers a computing platform to its customers. PaaS is a way to rent hardware, operating systems, programming language execution environments, storage, Web server, and network capacity over the Internet. The service delivery model allows the customer to rent virtualized servers and associated services for running existing applications or developing and testing new ones. In the PaaS models, cloud providers deliver a computing platform. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers.

SaaS

In SaaS, users are provided access to application software and databases. cloud providers manage the infrastructure and platforms that run the applications. SaaS is sometimes referred to as "on-demand service" and is usually priced on a pay-per-use basis. In the SaaS model, cloud providers install and operate application software in the cloud, and cloud users access the software from cloud clients. cloud users do not manage the cloud infrastructure and platform where the application runs. This eliminates the need to install and run the application on the cloud user's own computers, which simplifies maintenance and support.

Cloud applications are different from other applications in their scalability—which can be achieved by cloning tasks onto multiple virtual machines at run time to meet changing work demand. Load balancers distribute the work over the set of virtual machines. This process is transparent to the cloud user, who sees only a single access point.

SCM and Cloud Computing

The popularity of cloud computing has changed the SCM function also. According to Amies, et al., the purposes of SCM tools have become merged in some cases. The SCM tools themselves have become virtual appliances that can be instantiated as virtual machines and saved with state and version. The tools can model and manage cloud-based virtual resources, including virtual appliances, storage units, and software bundles. The roles and responsibilities of the actors have become merged as well with developers now being able to dynamically instantiate virtual servers and related resources [4].

As discussed earlier, in the cloud you can provision virtual machines, storage, and test data. However, setting up the instances manually in the cloud is not very practical. Alternatively, you can make use of automated provisioning tools to automate the entire array of provisioning tasks in the cloud. Automated provisioning is the ability to deploy an IT service by using predefined procedures that are carried out electronically without human intervention. Automated provisioning depends upon a predefined specification. This specification or procedures essentially constitute a declarative representation of how you want your system to be provisioned. The provisioning tools will execute the specifications and procedures.

In cloud computing, automated provisioning tools makes the task easy and error-free. Also, if one is using sophisticated tools for automated provisioning, the job does not need a highly skilled professional; anyone with a basic knowledge of the commands and the necessary permissions can do the job. Monitoring software can be used to trigger automatic provisioning tools when existing resources become too heavily stressed.

Cloud computing and SCM in cloud computing are still in their infancy. The number of automated provisioning tools, SaaS SCM tools, organizations moving to the cloud, and organizations leveraging their on-premises development environment with the on-demand cloud environment are all increasing. So cloud computing, software development, and SCM in the cloud all have a bright future and will continue to grow in quality and quantity as more and more innovations make tasks easier, error-free, and faster.

Hosted System Deployment

The traditional SCM solutions can be hosted off site. The system has the capability to access real-time data from anywhere without using complicated and costly remote access software. When you host your SCM system on an ISP server, then it becomes hosted SCM.

Two terms connected to the cloud and hosted environment are PaaS and IaaS. As discussed earlier, PaaS refers to the virtual renting of servers, hardware, storage, operating systems, and other data storage capacities. PaaS offers hardware to run your software package. IaaS is similar to PaaS but deals more with the components such as Internet connectivity and virtual desktops.

When you want to run your SCM system on a hosted environment, you use the PaaS and IaaS. In such cases, you own your software and allow someone else to provide the computer equipment to run it. The computer equipment resides at the hosting company's facility, and you access your software over the Internet. Just like an office or factory building lease, you can design the environment and have exclusive access to it; the building is maintained by the property owner or manager, but you are responsible for the use of your space and its contents. With a hosted computing arrangement, you specify and design the needed computer equipment (the environment), you have exclusive access to the equipment, the facility that houses the equipment is maintained by the hosting company but you are responsible for the software, the maintenance of your software, and the related data.

In the above mentioned scenario, it is possible to have different arrangements with regard to ownership of the computer equipment. In some cases, the hosting company will provide the computer equipment and all related maintenance. It will typically support the equipment and include the maintenance and support costs with your regular monthly fees. In other cases, you may purchase your computer equipment on your own and allow the hosting company to house it and maintain it for you.

Typically, software implementations, maintenance, and upgrades are as costly and challenging in a hosted environment as they are in an on-premises environment. Depending on whether or not the hosting company provides the computer

equipment or you buy it, the costs for housing and maintaining the equipment can be expensive, coupled with the upfront costs for purchasing software licenses and optionally computer equipment. In this model, you will have an ongoing, recurring cost for your hosting fees.

Advantages of a Hosted SCM System

The advantages of a hosted SCM system are listed as follows:

- You "own" the software and only pay once beyond the maintenance which is usually between 15% and 20% of the cost of the software. If you stop paying maintenance, the software will continue to work at the version you are on.
- Your data is in a very secure data center that may also have or offer multisite redundancy in case of disaster. Backups are made reliably, and you can connect from almost anywhere
- You can still bring your application back in-house, if you want, with little interruption. Most hosting centers use "virtual" servers like VMWare or Microsoft Hyper-V. This enables you to take your "server" and run it on your own physical hardware quickly.

Disadvantages of a Hosted SCM System

The disadvantages of a hosted SCM system are described as follows:

- The monthly hosting cost may exceed in-house costs over the long term, depending on variables such as other nonhosted solutions you have in house.
- If your office Internet connection goes down, you have no access to your system. Redundant Internet lines to your office can help alleviate this risk but will incur more costs.
- If you want to integrate other solutions to your SCM system, you usually have to have that solution supported and installed at the hosting center. as it is difficult for local applications to real-time-integrate with hosted applications.
- You need a fair amount of bandwidth, which will result in higher usage charges.

SaaS or On-Demand Deployment

SaaS is a concept in which you rent everything including your computing equipment and your software. SaaS specifically refers to software that is delivered on-demand over a network. The software itself is not licensed or owned by the end user. It is provided as a service. SaaS model offerings are typically smaller. This is normally best for organizations with limited complexity, size, and global presence. SCM system providers host SaaS models on their own infrastructures instead of locating the systems on-premises at the purchasing organization.

SaaS differ from traditional hosted services in three ways:

- SaaS technology is purchased on-demand, typically in per-minute or hourly increments.
- SaaS is elastic, with dynamic provisioning that lets users buy only as many services as they need.
- The provider completely manages the entire range of services, which means that users typically only need a computer and Internet connection to access them.

In the SaaS model, the vendor will have full control of the application and not the customer, whereas in the case of hosted system, your data and your application can be placed on the server that you control. SaaS provides instant gratification, taking mere minutes to be up and running.

In this model, you pay a monthly, quarterly, or annual fee and simply use "the system." Just like in a furnished rental office, you only pay a monthly fee, and there is nothing to own or maintain. You access the system through the Internet and have no responsibility for the computing environment including any maintenance to your system. Your system provider is responsible for upgrading your software and hardware, for making sure everything is working, and for expanding your computing resources as your needs increase. In this case, you simply sign in to your system and go to work.

The costs with a SaaS or rental model are typically consistent and predictable over a period of time. Most SaaS companies will charge a monthly, quarterly, or annual fee for access to the system, and there are no other costs to incur. In addition, just like the hosting model, you will have an ongoing, recurring cost as long as you use and access the system but your upfront investment costs are minimal.

Advantages of SaaS SCM Systems

The advantages of SaaS SCM systems are described as follows:

- SaaS solutions can be financially attractive as the initial investment is low.
- SaaS SCM solutions can be implemented with relative ease and integrated faster into the organization's day-to-day operations than their on-site SCM counterparts. In fact, most SaaS SCM solutions can be up and running in days, rather than months or even years, as may be the case with traditional on-site SCM software.
- Often, the services component of implementation and future upgrades is bundled into the monthly fee. It may be easier to swallow $200 per user per month than $150,000 for a new SCM solution. Even if the $150,000 is actually cheaper over 5–10 years, the cash savings is very attractive to some companies.
- With traditional licensed SCM software, organizations typically must wait for the next release to benefit from the latest features, upgrades, or security patches. In addition, the cost, complexity, and potential disruption of moving to a new on-site software version often cause some organizations to defer upgrading to the newest release. This, in turn, prevents employees from taking advantage of the latest productivity-enhancing tools consistently being added to the applications. SaaS SCM systems, however, eliminate this common

problem. Under this delivery model, the provider continuously and unobtrusively adds the latest features and upgrades, which means that users can be assured that they're actually using the latest technology.
- To be truly SaaS, it is likely the applications were recently developed. This gives many SaaS applications a "fresher" look and feel and more modern technology than older applications that were moved into the cloud or are hosted.

Disadvantages of SaaS SCM Systems

The disadvantages of SaaS SCM systems are described as follows.

- Although cheaper upfront, the monthly fees, can add up over time to be substantially more than an "on-premises" solution. SaaS vendors will typically overinflate the cost of in house IT and upgrades, and on-premises vendors will typically underestimate those costs. Do your own realistic analysis of the whole picture and look at it over 10 years, which seems to be the life of a major SCM system these days. SaaS can have additional fees for extra bandwidth and storage.
- There is nothing to keep an SaaS vendor from increasing the monthly fees a year or two down the road after you have invested time and money in implementing the system. With SaaS, if you do not pay, you lose access to your system, period. With on-premises, if you stop paying maintenance, your software continues to operate at the current version level.
- Some SaaS vendors have contracts that do not even allow you to retrieve you own data until they are paid in full. This possession of data can be a big sticking point once the lawyers start looking at the contracts.
- Organizations do not own the code in most SaaS models, have limited rights and control, and are at the full mercy of the vendor should the vendor decide to take a different product direction or find itself bankrupt.
- The total cost is still ambiguous. As more data and connections grow by the day with a deployment, changing SaaS vendors raises mounting complications with the advent of time. Migration plans across vendors or to on-premises solutions are unclear and complicated due to business processes latched onto vendor functionality and varying metadata and standards and process flows.
- As the SaaS model is still early in enterprise adoption maturity, most vendors still have customer-friendly policies. As more businesses move to the SaaS model, the deals favor the vendor. Unless rights are stated upfront today, buyers will lose leverage over time.
- When the interface is HTML/Web only, many solutions are slower for "heads down" data entry or are missing the richness of a traditional windows application such as right mouse click drill downs. If you bring up a defect list, many systems will show you 20 records at a time, and you hit a "next" button to browse the next set, whereas a premise/windows application can scroll through thousands of records quickly.
- Since SaaS vendors are starting up quickly and many are only operating due to venture capital or other equity money, their long-term survival is questionable as the inevitable consolidation occurs. If your SaaS vendor goes

bankrupt—even if it gives you a chance to get your data—it could take months to reimplement a new system. Can your business survive that interruption? With premise systems, if the vendor goes out of business, you can move to a new solution at your own pace.
- SaaS can be challenging when expanding to multiple projects or geographical locations in a large organization, and there may not be significant cost savings with an SaaS model in complex deployments involving sophisticated business processes.
- Most SaaS solutions have only a limited ability to customize, and integration of SaaS solutions with existing legacy systems can be very challenging.
- Regarding data security, the customers have to trust the vendor. So if the vendor's security infrastructure and policies are not sound and foolproof, then there will be security issues that can seriously compromise the organization's operations.
- The operational transparency is less in the case of the SaaS model, as customers do not have clear visibility into the system's health.
- SCM implementations are people projects and are difficult to manage. So the relative inflexibility of SaaS will force organizations to change their business processes, which can magnify the organizational change management challenges.

When an organization is implementing an on-premises SCM system, it can take effective measures to secure and safeguard the data and its security and privacy. However, in the case of an SaaS solution, these issues are handled by the solution provider or vendor. So, if you are considering an SaaS SCM solution, be sure to ask potential solution providers the following security-related questions:

- *What is your privacy policy?* Your solution provider should have a well thought out and documented privacy policy that is compatible and acceptable to your organization.
- *What level of security do you use to ensure the safety and integrity of critical data?* To safeguard your data on-site, your prospective solution provider should use a combination of intrusion detection system (IDS) and intrusion prevention system (IPS) products and apply antivirus at various network layers. It should also utilize deep-packet inspection (DPI) or an application level firewall technology that scans all levels of packet transmission. Finally, it should also use secure socket layer (SSL) or https-encrypted transmission to ensure Internet security.
- *Is your equipment housed in a state-of-the-art facility?* Your prospective vendor's data center should be secure and strong and capable of withstanding natural calamities.
- *Please describe your facility's physical security arrangements. Are they in place 24 hours a day, seven days a week, and 365 days a year?* The solution provider should have well-defined and robust security arrangements that are in place at all times.
- *Do you contract with an independent, third-party organization to conduct periodic external and internal vulnerability scans?* In addition to maintaining

an intrusion-response system and a prepared response plan, your prospective solution provider should frequently commission both routine and unannounced security audits.

- *How often do you back up data, and where are the backups stored?* A rigorous program of data backup and off-site storage in a secure location remote from its main data center should be in place.
- *Do you offer full hardware redundancy to avoid the negative consequences of a power failure?* The data center and backup location should have redundant power supplies, such as battery and diesel generator backups, to avoid the negative consequences associated with a power failure.
- *How do we ensure that our hosted SCM solution will remain affordable and viable as our business requirements and competitive environment evolve?* As the demand for SaaS solutions grows, the solution provider will gain more leverage and will be in a position to dictate terms and conditions. So what assurances can the vendor give that they will not take you for a ride when the situation favors them? The solution is to agree on the rates, the rate of increase in rates, and other costs and get these things in writing.
- *Does your staff include a highly qualified operations team that monitors the site 24 hours a day, 365 days a year?* Your prospective vendor should employ many certified security experts, including those with the preferred certified information systems security professional (CISSP) designation.

We have considered the different methods for deploying SCM systems. The beauty of modern SCM packages is that you have options to choose from during your SCM software selection. You can either deploy a solution by hosting it internally on your own servers (traditional SCM), or if you would rather not deal with the software, you can have it hosted somewhere else (SaaS). So which model is best? In many cases, there is no one best answer on whether or not to move your business application to the "Cloud." Your business circumstances, management reporting requirements, and operational needs all impact your decision. Changing business management systems is costly, time-consuming, disruptive, and not without risk. Having the ability to off-load much of the overhead and maintenance of on-premises computing provides businesses with more stable and reliable systems. However, there are some considerations that will help bring some clarity. They include the availability of internal IT resources, the SCM time horizon, and the company size. In deciding whether to use a SaaS platform, companies may take the following factors into consideration:

- *Simplicity:* In general, SaaS is simpler to deploy from a technical perspective. Because you do not need to purchase additional servers or physically install the software yourself, you can simply get started with a basic Internet connection.
- *Flexibility:* Because traditional SCM is installed on your servers and you actually own the software, you can do with it as you please. You may decide to customize it or integrate it to other software, for example. Although any SCM software will allow you to configure and set up the software as you

like, SaaS is generally less flexible than traditional SCM in that you cannot completely customize or rewrite the software. Besides, SaaS vendors are more likely to provide only one version, whether you need the upgrade functions or not, and since the software is delivered over the Web, you have no choice but to accept them.

- *Control:* Many companies find that they do not have as much control over SaaS software as they would like, relative to traditional SCM. This is especially true of midsized or large companies with well-defined business processes that cannot be changed to fit the software. Small companies can generally adapt their business processes to the software more easily than large organizations can.
- *Accessibility:* SaaS uses the Internet to access systems once deployed, and the systems are maintained off-premises. Since SaaS is entirely accessed through the Web, you are in trouble if the Internet goes down. Traditional SCM does not require Internet reliability, provided your users are accessing the software from inside your company's network.
- *Cost:* In general, SaaS can be deployed at a much lower initial cost and a lower total cost of ownership; companies can avoid expensive licenses and complex hardware and software infrastructure, and there is no technology maintenance needed. This type of cost-effectiveness can be attractive to smaller businesses, but the ongoing annual payment or monthly fees can be higher for SaaS because you are paying to use the software on a subscription basis. It can become costly as you grow and add employees to the system.
- *Integration:* Companies that have implemented traditional SCM typically have applications that run on the same platform. They have avoided tough integration issues and improved visibility into operations. Integration is a major issue for SaaS companies, which need to provide on-premises integration for their customers to integrate cloud applications with existing legacy applications. A company with SaaS will find it very difficult to integrate hosted software from a variety of vendors using middleware from yet another vendor.
- *In-house IT expertise:* Use SaaS when you lack internal IT resources. Businesses benefit from SaaS when they do not have IT resources to dedicate to installing and managing applications.
- *SCM time horizon:* Another consideration is the length of time your organization plans to use the system. For short-term deployments of small companies, SaaS is a better choice as it can provide lower costs, but for longer durations, the SaaS model will be less expensive.
- *Company size:* SaaS is normally best for organizations with limited complexity, size, and global presence. For large organizations with worldwide presence and complex business models, on-premises deployment is the best, as they will have their own IT departments and can enjoy data security and flexibility of operation.

Of the different SCM deployment methods, the traditional on-premises SCM is still the de facto choice. Nevertheless, other deployment options are being considered and are becoming popular. Advancements in areas like data security and Internet access will spur the growth of other deployment methods. Also, SCM vendors are

Table 19.1 SaaS Versus On-premises Choice Matrix

	SaaS	*On-premises*
Business process	Simple business process	Complex or unique business process
	Immature or undefined business processes	Well-established business processes
Employees	Low tenure	High tenure
Business model	Stable, little change	Volatile, constantly changing
Company size	Small to medium	Medium, large, or global
IT skills	Limited	Sophisticated
IT infrastructure	Little to none	Well-established
Integration with other systems	Little to none	Need for integration to other applications
Control	Little need or desire	Require control

offering SaaS and Web-based solutions, and this will help in reducing the total cost of ownership and time to implement the system.

When deciding which method to choose, make sure you understand the costs, responsibilities, limitations, and future obligations associated with the different models and then select a deployment method that is best suited for your organization. There is little doubt that SaaS SCM represents the wave of the future. SaaS offers some unmatched benefits including scalability, ease of upgrade, and lower IT administration needs. However, SaaS introduces inherent business continuity risks associated with the relinquishment of data control. Unless businesses choose to outsource the hosting of their on-premises systems, they can avoid these risks. Table 19.1 summarizes the factors that should be considered and which model to choose based on those factors.

Deciding between SaaS and on-premises systems will inevitably be one of trade-offs. The key to making an effective decision is to prioritize business requirements across an expected investment horizon. Only then will a company be in a position to make a value-maximizing SCM decision.

SCM Transition Strategies

Selecting and implementing a new SCM system and the process changes that go with it is unquestionably a complex undertaking. Regardless of a company's size and perceived resources, an SCM implementation is not something that should be approached without a great deal of careful planning. Among companies that have been through a less than fully successful SCM implementation, five reasons for poor results show up consistently:

- The implementation or transition strategy chosen was not the one suited for the organization.
- The implementation took much longer than expected.
- Preimplementation preparation activities were done poorly, if at all.

- People were not well prepared to accept and operate the new system.
- The cost to implement was much greater than anticipated.

The most important factor that decides the success of an SCM implementation is the transition strategy. This section discusses the various transition strategies, their advantages and disadvantages, and the suitability of each of them.

An SCM implementation strategy determines how the SCM system will be installed. Different companies may install the same SCM software in totally different processes. The same company may implement different SCM software in the same approach. There are three commonly used methodologies for implementing SCM systems. There are several transition strategies but most of them are variants of the two basic types, listed as follows:

- Big-bang;
- Phased.

These techniques focus on the strategy of how to make the transition from a legacy system to a new SCM system. SCM implementations all begin with a simple question: how do we make the transition from our legacy SCM system to the new SCM system? The selection of the transition strategy that is best suited for each organization is crucial as a wrong strategy can result in a failed or flawed implementation.

Three pillars—process, people, and technology—support any SCM implementation. Failure to use any one of these or failure to use them in the best possible manner can result in improper implementation. Understanding the relationships of SCM transition strategies between the process, people, and technology will assist the SCM implementers to better understand what type or combination of types of SCM transition strategy is best. We now consider each of these transition strategies in detail.

Big-Bang Strategy

In this strategy, companies lay out a grand plan for their SCM implementation. The installation of SCM systems of all modules happens across the entire organization at once. The big-bang approach, which promises to reduce the integration cost in a thorough and careful execution, dominated early SCM implementations, and it partially contributed to the higher rate of failure in SCM implementations.

Today, not many companies dare to attempt it. The premise of this implementation method is treating SCM implementation as the implementation of a large-scale information system, which typically follows the SDLC. However, an SCM system is much more than a traditional information system in the sense that the implementation of SCM continuously calls for the realignment of business processes. Many parties involved in SCM software systems are not IT professionals. SCM more than automates existing business processes; it transforms the business processes.

In the big-bang strategy, the company moves from the existing or legacy system to the new SCM system on a specific date. All the business functions performed in the legacy system across the entire enterprise are simultaneously transferred to the new legacy system during a period of one day or a weekend. The big-bang strategy

is seldom used and not often recommended by SCM vendors, systems integrators, and service providers. Many companies struggle in deciding whether the big-bang approach is the right choice for them.

One of the reasons given for not using the big-bang approach is that it consumes too many resources to support the go-live of the SCM system. High failure rates have plagued users of the big-bang approach, but high failure rates have also been associated with other strategies! Success in using the big-bang strategy comes with careful preparation and planning prior to using big bang. It is not a question of whether the big bang is a good approach for SCM systems. The success of the big-bang strategy depends on how well an organization plans and prepares itself prior to implementation.

Large-scale scientific and technical projects requiring mass coordination are often successful when using careful preparation and sound planning. Large and complex projects such as building very large structures like bridges, skyscrapers, satellites, space shuttles, fighter planes, and ships have been undertaken successfully in the past and also in the present. In all these projects, success has depended on the amount of planning that went into the design and resource allocation. However, most of these projects are far more complex than implementing an SCM system in a company.

With careful planning and preparation, companies using the big-bang strategy can be just as successful (if not more) as with any other approach. The big-bang strategy offers the following advantages if properly used:

- Its overall cost of implementation is less because no interface programs are required to communicate between the legacy system and the new SCM system.
- It eliminates all of the sequencing and decision-making of implementing one module at a time.
- It is well-designed for rapid implementations.
- It creates a strong central focus for all the SCM team members.
- It can avoid complex integration issues.

However, there are disadvantages also, including the following:

- Careful planning and preparation for the go-live consumes a large amount of time and requires a large financial expenditure.
- A bottleneck of critical resources such as lack of funds and nonavailability of professionals during the implementation can result in failed implementations.
- The recovery process, if something has gone wrong, is more difficult in this approach; it is a do-it-right-the-first-time project;

The consequences of a failed implementation can range from anything such as a huge financial loss to the company going out of business.

Phased Implementation

In the phased approach, the SCM system is implemented for one project at a time, and after that goes live, the implementation starts for the next project and so on. Modular implementation reduces the risk of installation, customization, and

operation of SCM systems by reducing the scope of the implementation. The successful implementation of one project can benefit the overall success of an SCM system.

Interface programs are common in their use for the phased approach or any situation that contains phasing. These interface programs are required to bridge the gap between the legacy SCM system and the new SCM system until the new SCM system becomes fully functional. A good example is where project A goes live on the new SCM software while project B still remains active on the legacy SCM system.

The phased approach is closely related to the big-bang technique. The advantages of this approach are described as follows.

- It allows companies to implement one project at a time before another is attempted.
- Many companies feel more comfortable taking this steppingstone approach.
- The total number of resources needed at any one given point in time may be less.
- Additional flexibility may also be gained in the scheduling of people.

There are disadvantages too. These are described as follows.

- This approach consumes a large amount of technical resources because of the conversion and interface programs that are required between the two SCM systems.
- The overall cost and time to implement is usually higher using this approach.
- Higher turnover rate can also be expected among key SCM team members because of the lengthy durations that generally accompany this approach.

Choosing a Strategy

The factors that could cause a company to choose one SCM strategy over another are technical resource availability, number of users, consultant availability, structure of the SCM team, deadlines, reliability, and hardware resources.

If you are going to develop a global process model for your company, then a big-bang approach is extremely beneficial. You can concentrate all needed expertise within one location and build a so-called competence center. Once you develop a rollout kit, containing such items as work instructions, training materials, plan of approach, detailed planning, templates, example deliverables, standard or customized software components, parameter setting, a chart of accounts, and hardware requirements, you will see dramatic lower implementation costs and much quicker implementation and stop local sites from reinventing the wheel. One factor that impacts employee buy-in is the determination of how exactly the SCM system will be implemented. The big-bang approach implies that the SCM system goes live for all projects at the same time. This approach forces all employees to use the new system with no opportunity to fall back on the old and often results in temporary business disruptions. The phased approach in which companies implement the SCM for one project and only after its successful completion move on to the next

project can help avoid business disruptions, but it may also create some resistance to change within departments.

Summary

This chapter deals with the different deployment models and the transition strategies of SCM implementation. The deployment models covered in this chapter are on-premises deployment, hosted deployment, and SaaS or on-demand deployment. The two transition strategies that are discussed are the big-bang and incremental approach. The discussion of each topic covers their advantages and disadvantages and details factors to consider in choosing a deployment model or transition strategy.

References

[1] Wikipedia, Cloud Computing, http://en.wikipedia.org/wiki/Cloud_computing.

[2] Evolven, Cloud Configuration Management Solution, http://www.evolven.com/solutions/cloud-configuration-management.html.

[3] Mell, P., and T. Grance, The NIST Definition of Cloud Computing, National Institute of Standards and Technology, U.S. Department of Commerce, September, 2011.

[4] Amies, A., et al., "Develop cloud applications with Rational tools," IBM developerWorks, June 5, 2012, (http://www.ibm.com/developerworks/cloud/library/cl-rationaltools/index.html).

CHAPTER 20

Source Code Repositories

Overview

A source code repository or source code hosting facility is a place where large amounts of source code are kept, either publicly or privately. They are often used by multideveloper projects to handle various versions and handle conflicts arising from developers submitting conflicting modifications in an organized fashion. They help developers submit patches of code in a structured way. Often these Web sites support features like sample code, code review, version control, bug tracking, release management, translation systems, mailing lists, support forums, and wiki-based documentation.

Source code hosting Web sites have changed the way software development is done. It has affected small-scale projects by individuals and start-up companies, as well as massive open-source projects involving hundreds of developers and thousands of testers and quality assurance personnel. Since these hosting Web sites have features required for high-quality software development, the software development at those sites happens in an orderly and structured manner and follows time-tested best practices. Also, the user community and the support forums help the developers who are stuck, thereby moving the projects forward. Due to the open and collaborative nature of most of the hosting Web sites, many resources like buildpacks (collections of scripts for compiling applications) and sample code are freely available, thus improving the productivity as well as the quality of software development. The popularity of these Web sites are increasing and will continue to increase as the software development community—open-source projects, small firms, start-ups, and individuals—has found this model of development cost-effective (often free, in fact), efficient, and more productive.

One of the major concerns in using code-hosting Web sites is that the provider might close shop, and you will lose all your source code and data. However, with the increasing popularity of distributed version control systems, you always have a full copy of your code and version history, and as long as you have a good backup system, you can migrate from one provider to another easily. Most of the repository providers have tools that make these migration easy and painless.

Software Development in a Code Repository

To develop software or collaborate on software development, you should first create a repository or "repo" on the hosting Web site. After creating your repository,

you must download and install the desktop client of the hosting software onto your local computer. You then download the source code of the project that you want to enhance or modify and work on it in your local development environment. The desktop client monitors this development and manages the uploading and downloading from and to the repository, performing such tasks as committing the changes and doing version control on the local machine.

Once you are satisfied with your work and have tested it, you submit it to the Web site for review and, if it is accepted, merge it with the master branch of the project code. If you are working on a development project, you will download the specifications the project leader has created and assigned to you and write the source code for that. Here also, once you have finished coding and testing, you submit it to the repository for reviewed and discussion, and if it is found acceptable, it will be pulled into the project codebase.

The owner of the software under development or the project maintainer will create a repository for the project. There are two popular models of collaborative development—the fork and pull and shared repository models. The fork and pull model is mainly used for enhancing a project or creating a new project using the source code of another project as the starting point. The shared repository model is mainly used by small teams and organizations for software development.

The fork and pull model lets anyone fork an existing repository and push changes to their personal fork without requiring access be granted to the source repository. This model is mainly used for helping someone with software development (e.g., bug fixes and enhancements) or creating new software from an existing code base. In this model, you should create a fork or clone of the existing project in your repository. A fork is a personal copy of another user's repository that lives in your account. When contributing to an open-source projects you will have a fork of the source code repository in your local development environment. Here, you can make your changes and commit them locally to create a revision history, allowing changes to be tracked and easily rolled back if required. Changes committed locally can then be submitted to the upstream project for inclusion in the next release. Once contributors are satisfied that their changes are worthy of consideration by the project maintainers, they raise a pull request. Pull requests are proposed changes to a repository submitted by a user and accepted or rejected by a repository's maintainers.

Forks allow a developer to make changes freely to a project without affecting the original. In the case of collaboration, once you make changes or enhancements, the changes must be pulled into the source repository by the project maintainer. This model reduces the amount of friction for new contributors and is popular with open-source projects because it allows people to work independently without upfront coordination. Forks remain attached to the original, allowing the developers to submit a pull request to the original's author to update with your changes. Pull requests are useful in the fork and pull model, because they provide a way to notify project maintainers about changes in your fork. You can also keep your fork up-to-date by pulling in updates from the original.

The shared repository model is more prevalent with small teams and organizations collaborating on private projects. Everyone is granted push access to a single

shared repository, and topic branches are used to isolate changes. Pull requests are used in the shared repository model to initiate code review and general discussion about a set of changes before they are merged into a mainline or master branch.

So the code hosting Web sites make the software development process—maintenance, enhancements, as well as new development—efficient, effective, and productive. Things like version control and change management are easily taken care of by the software. For example, a CR in a traditional setting would require the CCB to meet and then approve the CR to come into effect, whereas in this scenario, the pull request will trigger a discussion at the end of which the change is reviewed and, if accepted, is merged without a physical meeting. Working at one's own convenient time, working from anywhere in the world, and working from different time zones are some other advantages of this model of software development, and all these advantages are realized without sacrificing the quality of the software that is developed.

How Will Repositories Help Software Companies?

Source code repositories are a great help for software development. Irrespective of whether you are an individual, a group of three or four developers, a start-up, a small company, or a medium sized business, you can make your software development process more efficient, effective, and productive using the source code repository. The quality of the code will improve because of the collaborative model of development.

When you create software in a Web-based source code hosting repository, there are many things already taken care for you. You are not beginning from scratch; you have all the necessary tools and help available at your disposal. It is like moving into a furnished apartment instead of an empty one. Repositories offer many features such as version control, bug tracking, chat rooms, discussion groups, and support forums. You and your team members can make use of all these features to create better software. Since the repositories operate on a collaborative-community model, you can ask for help from others if you are unable to find a solution for your problem. Experts in different technologies in the community will answer your queries, and you can have discussions to find the best solution before going ahead. This will help in improving the quality of the code you are creating.

Another advantage of these repositories is that you can avoid reinventing the wheel. Since there are many developers and groups sharing the same space with you, most of the things you want, like code snippets, scripts, and validation functions will be available for immediate use, thus cutting down on development time.

Flexibility is another great feature of this development environment. Flexibility includes flexibility of time and location. You can work from anywhere in the world and at any time of day. Since these Web sites work in the asynchronous mode, you do not have to have face-to-face meetings with your colleagues. You can work when you are most productive and yet have meaningful interaction with your team members, thereby making you flexible without compromising on quality. The chat sessions and support and discussion forums can handle an asynchronous mode of communication. So if you send a request or query to your team member, he or she

will answer when it is convenient. This type of time and location independence improves work satisfaction and results in happy employees.

So, with their feature-rich development environment, community support model, and flexibility, source code hosting Web sites help software companies without the time and resources to set up such infrastructure to develop high-quality software. With these repositories, you can start development within minutes after you sign up. Most Web sites offer free facilities for open-source projects, individuals, and small teams and charge a nominal fee for organizations that want private repositories. This is a real blessing for small companies and start-ups, as they do not have to invest too much capital in setting up the infrastructure.

Features Available at Source Code Repositories

Source code repositories have many features that make tasks such as coding, code review, code enhancement, bug tracking, bug fixing, and version control easy, fast, and painless. Some of the major features are described as follows.

- *Version control:* All the repositories usually have two or three version control systems enabling you to choose the one that you prefer. These version control systems help you in such tasks as creating a version history, branching, merging, and forking (or cloning). Accordingly, you will be automatically using version control from day one, thus eliminating the chaos and confusion caused by their absence.
- *Bug tracking systems:* Bug tracking or error tracking is another feature of these repositories. These systems help in creating bug reports, evaluating them, fixing them, getting the bug-fixed version reviewed, and then merging the bug-fixed version into the main branch.
- *Sample code:* In repositories, sample code, code snippets, scripts for running applications, and reusable functions created by the development community are available. You can use these as is or improve them so that people can have better code. This availability of reusable code and scripts and the facility to modify and improve them and then make the better versions available to the development community is the beauty of the collaborative development model.
- *Discussion forums:* The discussion forums are available for all topics that relevant to the developers. One can make queries, solicit opinions and suggestions, and ask for help in solving problems in these discussion forums. The members of the forum include experts in each field and experienced and seasoned veterans who have seen it all and are willing to help and mentor newbies. Developers can make use of these forums not only to improve the code but also for their own professional development. Getting advice from the gurus of the field is not only inspiring but also a way to learn from their experience.
- *Help and documentation:* The Web site provides detailed on-line help and exhaustive documentation so that beginners can start without an expert to help them. Most of the repositories are easy to manage, but doing advanced functions like forking, branching, raising a pull request, and merging, might

be difficult to do without some assistance. The help and documentation provides step-by-step instructions (with figures) on how to perform each task with examples. This will save a lot of time for newbies. Also, they will learn the correct method of doing things from the very beginning, thus developing good working habits.

The preceding are the most important features available at repositories, but there. are many more. Visit any hosting site to learn about additional features.

Factors to Consider When Choosing a Repository

There are several factors to take into account when choosing a repository. Some of these are outlined in the following:

- Does the repository have a pricing package that is suitable for you?
- Does the repository support the programming language or languages of your choice? Different repositories support different programming languages. Choose a repository that supports the language that you are using.
- Does the repository offer database support? If your project needs a database, then choose a repository that offers a database.
- What functionality do you need now? Depending on your project you might need a version control system, including a Web interface for such things as on-line code-browsing, mailing lists, list management and archives, bug trackin, software package hosting and publishing, statistic reporting, support and discussion forums, help and documentation, project and release management, and access control. You should consult with the feature set offered by the repositories and make sure that you are getting all you need.
- How easy is it to upgrade to additional functionality in the future?
- Does the repository have your preferred version control and bug tracking systems?
- Does the Web site support both public and private repositories?
- How easy is it to integrate other things you run separately (e.g., a Web site) with the repository?
- How good is the support for your integrated development environments (IDEs) of choice?
- Is there support for authentication systems such as OpenID or SSH keys?
- Does the Web site host projects that are similar to yours?
- Is the upload and download speed acceptable?
- How easy is it to backup the entire repository?
- How established and stable is the repository?
- How good is the user support?
- How much effort do you have to put into repository maintenance?
- What are the service-level agreements for uptime, downtime, time to fix outages, and bandwidth?
- How easy will it be to transfer not just your code, but your community, to a new site in case the repository is closing shop?

Advantages and Disadvantages

This section discusses some of the advantages and disadvantages of source code repositories.

Advantages

The advantages of software code repositories are desribed as follows:

- The initial investment is quite low and will be a financially attractive option especially for individuals, small teams, start-ups,s and small companies.
- The setup time is only a few minutes, so you can be up and running within minutes after you sign up with a service provider. This will help developers to start working from day one.
- The learning curve is very small, and the developers in your team or company can become proficient within a matter of hours.
- The development environment uses time-tested best practices, and hence developers will be using these best practices, thereby improving the quality of the software created.
- The development model is very flexible. Developers can work from any location and at any time. This location and time independence gives developers a lot of flexibility and will go a long way in improving their job satisfaction and happiness. Satisfied and happy employees tend to be more productive and produce high-quality work.
- The collaborative nature of the work environment will benefit the developers as they can take the help of more experienced and knowledgeable colleagues when they face a problem or have queries.
- The availability of such things as reusable code snippets and scripts will improve the productivity and quality of the software created.
- The features of the version control and bug tracking systems combined with the discussion forums eliminate bureaucratic processes like CCB meetings.

Disadvantages

The disadvantages of software code repositories are detailed as follows:

- Even though the monthly or usage fee is low, the service provider can increase the rates at any time. Such an increase can have an impact on financial planning and projections.
- The service provider can close shop at any time, and if you do not have a foolproof backup strategy, all your data can be lost. If you have a backup, you have the option of migrating to another provider, but sometimes the transition can be rough.
- The source code repositories are still early in enterprise adoption maturity, and most vendors still have customer-friendly policies. As more businesses move to these Web sites, the deals will start favoring the vendor.

Summary

A source code repository or source code hosting facility is a place where large amounts of source code are kept, either publicly or privately. These Web sites support features like sample code, code review, version control, bug tracking, release management, translation systems, mailing lists, support forums, and wiki-based documentation.

Source code hosting Web sites have changed the way software development is done. It has affected small-scale projects by individuals and start-up companies, as well as massive open-source projects involving hundreds of developers and thousands of testers and QA personnel. This chapter reviews the working, features, advantages, and limitations of source code repositories.

Selected Bibliography

Hong, N. C., *Choosing a Repository for Your Software Project*, Software Sustainability Institute (http://software.ac.uk/), 2013.

Wikipedia, *Comparison of open-source software hosting facilities*, http://en.wikipedia.org/wiki/Comparison_of_open_source_software_hosting_facilities.

CHAPTER 21

Implementation Challenges

Introduction

There are few information systems whose design and implementation challenge a company like SCM. Done right, a new SCM implementation can dramatically improve the software development and maintenance processes. But when an implementation fails—or takes a prolonged and arduous course—huge amounts of money and effort may be misspent. This chapter discusses the following challenges faced before, during, and after an SCM implementation:

- Inadequate requirements definition;
- Resistance to change;
- Inadequate resources;
- Lack of top management support;
- Lack of organizational readiness;
- Inadequate training and education;
- Inaccurate expectations;
- Poor package selection;
- Poor project management;
- Customization issues;
- Poor communication and cooperation;
- Data quality costs;
- Hidden implementation costs;
- Improper operation or use.

The frequency of such challenges has increased over the last few years, suggesting that companies are having increased difficulty implementing their SCM software. Regardless of the size or needs of an individual company, strong organizational risk mitigation and change management can address many of the issues cited above. Processes such as project planning, resource deployment, segmented communications, targeted training, and strong data conversion plans serve to minimize the negative effects of change, decrease the durations, and increase the success of implementations. Because of the resource and staffing constraints most small- or medium-sized businesses face, it is critical that they recognize the great impact that organizational change efforts—both in the executive suite and among end users—have on implementation success. Regardless of which software package is chosen, a company that does not devote time and effort to ensuring that its staff is aligned and trained and that its leaders are clear on the parameters needed for SCM success such as the

project priority, timeline, and staffing needs is apt to see an SCM project stretch well past the initially projected time frame.

Implementation Challenges

We now examine implementation challenges in detail. SCM implementations are more likely to fail, be delayed, cost more than forecast, or fail to deliver full functionality than they are to succeed. It is important to be aware of how SCM as a technology has evolved, its strengths and weaknesses, and the nature of important implementation challenges. Further, it is important to understand how SCM and legacy system metadata can be used to simplify implementation and help organizations best plan for this exciting new challenge. When considering SCM as a potential solution to specific organizational challenges, major considerations include solving the implementation challenges. The following sections examine implementation challenges in detail.

Inadequate Requirements Definition

Inadequate definition of requirements is one of the major challenges faced by the implementation team. The requirements definition should clearly specify the issues and problems that the SCM system is supposed to solve and the additional capabilities expected from the system. If the requirements are properly specified, then the implementation team can go about their job, including selection of the SCM package that is best suited to meet these needs and determining the areas where customization is needed and the functions where the organization's business processes should be changed. All these are important for the successful implementation and operation of the SCM system. Failing to provide these (which is the management's responsibility) could result in problems such as selection of the wrong SCM package, unnecessary customization, and lack of employee retraining, all of which can result in the failure of the SCM implementation.

Resistance to Change

Implementing an SCM system is a change, and it is human nature to resist change. So any SCM implementation will face some amount of resistance. Users will be skeptical about the new system. However, for an SCM implementation to succeed, the cooperation of everyone involved is an absolute necessity. SCM is first an attitude, then a system. So, if employees are not convinced of the importance of SCM and the benefits of using an SCM tool or system they will not be fully cooperative, which can result in the failure of the system. It is very important, therefore, that users be won over before implementing the system. Forcing the system on unwilling people will only harden their resolve to revolt.

A primary reason for the resistance is ignorance. People always have a lot of misconceptions about SCM—including that it will increase workload or hinder creative work. However, if the SCM implementation team, backed by management,

spends a little time and effort educating users about SCM and how it will help the company and the users, user resistance can be reduced—if not fully eliminated.

Another method of reducing resistance is by creating champions. One of the most efficient ways to transition to new technology is to find a well-respected potential user of the technology. Train the user on the process and the technology, have him or her evaluate the technology, and encourage him or her to champion the merits of the technology to coworkers and management. The champion becomes the expert user, facilitator, and trainer of the tool. There will always be people who adapt to change slowly and maybe even begrudgingly; do not look to them to be your champions. Instead, look for people who are the first to embrace change and adopt new technology and who are always looking for ways to do things better. That is the kind of person you want for a champion.

Inadequate Resources

SCM implementation is a very costly affair that requires a variety of resources—including money, people, software, and hardware. It is unlikely a company's management will support the idea of unlimited funding for an SCM implementation project. Accordingly, a budget, based upon an estimate of the likely costs, needs to be established. There are many items that will be missed during the preparation of the budget but will consume money during the implementation. The long implementation period will escalate many costs. Another resource that is always in short supply is skilled and motivated personnel from the organization. Getting the right people with the necessary skills, aptitude, and enthusiasm is one of the most difficult tasks faced by SCM implementation teams. These inadequacies in resources can create many challenges to the SCM implementation.

Lack of Top Management Support

The commitment of top management to the diffusion of innovations throughout an organization has been well-documented. In particular, early in a project's life, no single factor is as predictive of its success as the support of top management. The roles of top management in IT implementations include developing an understanding of the capabilities and limitations of IT, establishing reasonable goals for IT systems, exhibiting strong commitment to the successful introduction of IT, and communicating the corporate IT strategy to all employees. Research on project failures shows that project cancelations occur when senior management delegates progress monitoring and decisions at critical junctures of the project to technical experts. The importance of top management support is instrumental in the success of all SCM implementations. So, going ahead without solid backing from top management is a sure recipe for disaster.

Lack of Organizational Readiness

The main challenge in the successful implementation of SCM system is the preparedness of the organization for a new system of functioning. Management should make

sure that the organization, the work process, and the employees are amenable to adapt to the SCM system.

Inadequate Training and Education

The role of training to facilitate software implementation is well-documented in the MIS literature. Lack of user training and failure to completely understand how enterprise applications change the software development and maintenance processes frequently appear to be responsible for problem SCM implementations and failures. SCM projects appear to have a six-month learning curve at the beginning of the project. At a minimum, everyone who uses SCM systems needs to be trained on how they work and how they relate to the software development and maintenance process early on in the implementation process. Although many companies use consultants to help during the implementation process, it is important that knowledge is transferred from the consultant to internal employees. Companies should provide opportunities to enhance the skills of employees by providing training on a continuous basis to meet the changing needs of the business and employees.

Inaccurate Expectations

Information system failure has been defined as the inability of an IS to meet a specific stakeholder group's expectations, and successfully managing user expectations has been found to be related to successful systems implementation. The expectations of a company may exceed the capabilities of the system. SCM systems may fail to meet expectations despite positive contributions to the organization if the systems are oversold by the vendor. Careful deliberation of success measurement as well as management of expectations by the implementation manager of SCM projects are important factors. Management of expectations has an impact through all stages of the implementation life cycle.

The first thing to realize when considering SCM is that inaccurate expectations are the norm. Most SCM implementations today result in cost and schedule overruns. These outcomes are due to a lack of understanding of SCM implementation complexities. Routinely, the cost of implementing and the time required to implement are underestimated, while the scope of what organizations are able to implement are overestimated.

Poor Package Selection

Selecting a good SCM solution provider is another challenge. You must analyze the capabilities of the SCM service provider and make sure that the provider has the capabilities and the expertise to provide you with a good solution. The choice of the package involves important decisions regarding budgets, time frames, goals, and deliverables that will shape the entire project. Choosing the SCM package that best matches the organizational information needs and processes is critical to ensure minimal modification and successful implementation and use. Selecting the wrong software may mean a commitment to architecture and applications that do not fit the organization's strategic goal or business processes.

Poor Project Management

While many in the IS business consider project management an oxymoron, its importance in IT projects is well-documented, and numerous methodologies and management tools exist. Project management activities span the life of the project from initiating the project to closing it. The contingency approach to project management suggests that project planning and control is a function of the project's characteristics such as project size, experiences with the technology, and project structure. The vast combination of hardware and software and the myriad of organizational, human, and political issues make many SCM projects huge and inherently complex, requiring new project management skills. Specifically, proper management of scope is critical to avoid schedule and cost overruns and necessitates having a plan and sticking to it. A project scope that is too broad or ambitious can cause severe problems. Customization increases the scope of an SCM project and adds time and cost to an implementation. The minimal customization strategy, which allows for little if any user suggested changes and customizations, is an important approach for managing the scope of an SCM project. The high implementation risks of SCM projects imply the need for multiple management tools such as external and internal integration devices and formal planning and results controls.

Customization Issues

Because SCM attempts to permit organizations to capitalize on planned information sharing and cost avoidance, they make sense when existing organization procedures and data structures can be successfully adopted to match those implemented by the SCM. Most organizations discover how compatible the new SCM is with their existing systems and processes when they turn on the new system. When they realize that the SCM differs from the system it is replacing, organizations are faced with the possibility of either customization or tailoring the SCM. There are two basic choices for customization:

- Modifying the SCM to match the organizational processes or data structures;
- Modifying the organizational processes or data structures to match the SCM.

Choices 1 and 2 require an understanding of SCM and organizational processes and data structures. Most organizations approach the customization or tailoring decision without the proper information required to reach a good decision.

Poor Communication and Cooperation

Communication is the oil that keeps everything working properly. Communication is essential within the project team, between the team and the rest of the organization, and with the client. Poor communication between implementation team members and other organizational members has caused many implementations to fail. A key factor for the successful implementation of SCM systems requires a corporate culture that emphasizes the value of sharing common goals over individual pursuits and the value of trust between partners, employees, managers, and corporations. As SCM

systems are cross-functional and above departmental boundaries (e.g., development, maintenance, and QA), the cooperation and involvement of all involved is critical. SCM's potential cannot be leveraged without a strong coordination of effort and goals across business and IT personnel.

Data Quality Costs

A data management study performed by Price Waterhouse Coopers [1] revealed two troubling facts about organizational data quality:

- Only 15% of companies are very confident of the data received from other organizations;
- Only one in three companies are very confident about the quality of their own data.

Poor quality data input can be fatal to SCM projects. Just imagine the lack of confidence that the new system engenders if it delivers bad data more easily than the legacy system.

Hidden Implementation Costs

There are many items that are missed or that will consume more money than than allotted in the implementation budget. These cash shortages will hinder a smooth implementation, and if the company does not have sufficient reserves to bear the additional expenditure, the implementation will have to be left incomplete. Chapter 17 discusses the hidden costs of SCM implementation.

Improper Operation or Use

The most crucial challenge is the optimal utilization of the SCM software solution. You can have the best SCM solution implemented, but if its resources are not utilized to the fullest, the whole initiative goes to waste. However, if it is deployed appropriately, SCM solutions can create dramatic changes in your business performance.

Summary

There are few information systems whose design and implementation challenge a company like enterprise resource planning. Done right, a new SCM implementation can dramatically improve business processes. However, when an implementation fails—or takes a prolonged and arduous course—huge amounts of money and effort may be misspent. This chapter describes the challenges faced before, during, and after an SCM implementation.

The main challenges of an SCM implementation include inadequate definition of requirements, resistance to change (lack of buy-in), inadequate resources, inadequate training and education, lack of top management support, unrealistic

expectations of benefits, miscalculation of time and effort, poor communications, software business process incompatibility, poor project design and management, and poor SCM package selection.

Reference

[1] PriceWaterhouseCoopers, Data Quality 2004 Survey, PriceWaterhouseCoopers (www.pwc.com), November, 2004.

CHAPTER 22

SCM Operation and Maintenance

Introduction

Most companies treat SCM implementation as projects, with the assumption that someday these projects will end. And they are right; the implementation project will end, but the SCM system cannot end with the implementation. In fact, once the implementation phase ends and staff have started using the SCM system, the real benefits of the SCM will be seen. An SCM system is not a project; it is a way of life. No organization can say, "We're finished," and few ever will. There will always be new modules or features and versions to install, new persons to be trained, new technologies to be embraced, and refresher courses to be conducted. Even if an organization can declare final victory on the implementation of SCM, it will need more time to gain real business value from the SCM system. So SCM implementation requires a lifelong commitment by the company management and users of the system.

Ideally, when the SCM system goes live, people switch from their old practices to those required by the new system. If everyone is properly trained, then each person will know what he or she has to do when the new system is in place—the operation and maintenance (O&M) phase. If the processes have been properly tested, then operations will progress smoothly. If the data migration has been properly handled, then the data will make sense to those using it. The success of this phase is measured by the lack of problems. However, the implementation is not finished. Work then begins on realizing the benefits of the new system. If problems are experienced during the O&M phase, mechanisms need to be in place to deal with them.

When problems arise there should be a problem response mechanism that deals with them and that everyone is aware of. This mechanism should be simple and provide a means for tracking progress in resolving the problem. When a problem is identified, it should be reported to the person assigned with responsibility for coordinating problem resolution. This usually will be the project manager. Initial investigation will reveal whether the problem is readily resolved or requires further investigation or whether it is something that needs to be transferred to the vendor for resolution. Details about the problem are required, including a description of what was being done at the time the problem arose. The problem coordinator will establish who is going to deal with the problem and keep track of progress in resolving the problem. Readily resolved problems can be closed off quickly. The rest need to be monitored to ensure that they are being addressed. If this sounds a lot like the issues covered earlier concerning software problems, you are correct. To go a step further, quite often a company will use its existing software problem reporting

mechanism to deal with SCM system problems and issues. The benefit here is that nothing new has to be developed to deal with SCM-specific problems and issues. This is another benefit of the SCM system.

The permanent nature of the SCM system has numerous implications. The following sections discuss some of them. SCM implementation is just the beginning. For any organization to succeed and reap the benefits of the SCM system, it has to take action while keeping in mind the permanent nature of the system.

Employee Relocation and Retraining

One problem the SCM implementation can present is what to do with the existing SCM teams. This is applicable only in cases where an organization had large SCM teams and decided to introduce an SCM tool. The SCM tool will automate most of the SCM functions, and many jobs in the SCM team will become redundant. So the company should have a plan to relocate these people whose jobs are taken over by the SCM tool. However, the tools will also create new job openings as they eliminate old ones. The complexity of today's SCM tools makes it necessary to have specialists to manage and maintain these tools. Today's tools need database administrators, system administrators, operations personnel, and build and release managers, among other professionals. So the team members who have lost their jobs due to the introduction of the SCM tool can be given training and assigned to these jobs.

The development of new processes will result in the emergence of new job descriptions. Automation of manual tasks and the creation of new tasks make this inevitable. However, people tend to be resistant to change. Thus, human resource personnel need to be involved at an early stage in the implementation. The implications relating to changes in job descriptions need to be handled in an agreeable and friendly manner. Throughout, the benefits of what is being pursued should be promoted as well as the well-being of those affected. In organizations where change is a pervasive feature of life, this should not be a problem. In organizations where change is unknown, this has the potential to become a big issue. Handled carefully, everyone should emerge smiling.

Organizational Structure

Most organizations create implementation project offices and appoint project managers with the assumption that the project will end and life will go back to normal. However, it will not. Accordingly, organizations need a new organizational structure that reflects the ongoing need for SCM-related activity—not a project office. "Sponsorship" is one example of the need. Many companies appoint senior executive sponsors for implementation projects. Their expectations are probably that these executives will go back to their responsibilities once the installation is over. However, many companies do SCM implementation on an incremental basis. That is, they install the core modules first and then the additional modules until full SCM

functionality is achieved. Who is going to oversee those changes and ensure that they fit with the rest of the business? If the executive sponsor is temporary, who will ensure that the system and the business evolve hand-in-hand? The company should assign a person who is willing to take the ownership of the SCM system on a long-term basis.

Roles and Skills

The post-SCM organization will need a different set of roles and skills than an organization with less integrated systems. At a minimum, everyone who uses these systems needs to be trained on how they work, how they relate to the business process, and how a transaction ripples through the entire company whenever they press a key. The training will never end; it is an ongoing process. New people will always be coming in, and new functionality will always be entering the organization.

Many companies use consultants to help with the implementation process. This in itself is not a bad idea, but the problem is how most companies use consultants in SCM implementations. They do not transfer knowledge from consultants to internal employees. Because these systems are going to be around for quite some time, it is very important for company employees to have good knowledge (as good as the consultants) about how these systems work and how they can be configured to fit the organization. The person in charge of the implementation must make sure that consultants allow employees to work side-by-side with them on the implementation project, and before they leave, tap the most knowledgeable consultants on long-term system evolution issues.

In every business function and department that is affected by SCM, organizations will need one or more people who know the system and its relationship to the departmental processes. It is these people who have to save the system in the early days after system installation. It is these people who have to guide, motivate, and help their colleagues by working with them. They will answer questions, find needed workarounds, and let you know what is working and what isn't. These people will be the SCM team representatives—the champions—in each department. These people should have a dual reporting relationship with their managers and the SCM Manager. It is also useful to convene meetings of these people once in a while so that they can share knowledge and compare notes.

Knowledge Management

It is imperative that the knowledge and experience that is gained during SCM implementation and after is captured on an ongoing basis and made available to all. So when somebody encounters a problem, he or she can look up the knowledge base to determine if such a problem has occurred before. When new problems are identified and solved, they should be added to the knowledge base. Thus over a period of time the knowledge base will provide answers to most problems. In this way, even if a key employee leaves, the knowledge will remain with the company. A

number of companies have successfully implemented this strategy through a FAQ interactive database.

SCM Tools and Technology

Once SCM tools are introduced, the way in which the companies conduct SCM will change. With SCM tools, automation and new technologies will arrive. Companies should make it a point to familiarize users with these technologies and find ways to motivate them to use them. For example, most SCM tools can send notifications to the CCB members regarding a CR that was submitted. The CCB member can see the details of the CR and then query the SCM database to analyze its implications. So the CCB members can send their replies almost immediately. For this to happen, however, the CCB members should make use of these technologies. One member abstaining from this process can delay the CR disposition. So the company should have a plan to train and then motivate its employees to get the best out of the new features and facilities that are available to them.

Most systems can be configured to have an escalation mechanism—a mechanism that can escalate the issue to a higher authority if something does not happen within a specified period of time. For example, the system could be configured to send a mail notification to the supervisor of a CCB member if that member has not replied within a specified period. In such cases, senior managers should find out why the person is not using the technology and take the necessary steps to get him or her involved. Many people are dazzled by the technology or are afraid to use it. These fears should be alleviated to facilitate the proper functioning of the SCM system and to get the maximum benefit from the system.

As we have seen, the success of an SCM system is not primarily dependent on the sophistication or features of the tools that are installed. It is the attitude and cooperation of the people—the users—that allow SCM systems to deliver the quality and productivity improvements of which they are capable.

Review

Once the implementation is complete, and the organization has started to use the system and has reached a stable state, a review of the system must be performed. The review, which aims to answer the following questions, provides an opportunity to learn from the implementation.

- Does the software do what is expected of it?
- What are the outstanding or emergent issues?
- What can be learned from the implementation?
- What could have been done better? How can this be used in future?
- What timescale or budget is required to deal with the remaining issues?
- Has the project been a success? This is particularly invaluable if there are successive phases involving the introduction of additional functionality.

The review should also elicit feedback from users and managers. This feedback may reveal issues that can be readily addressed such as the provision of extra training or the modification of existing procedures.

Operation of the SCM System

We have seen the postimplementation scenario of the SCM system. It is during the O&M phase that the SCM system delivers the full benefits of which it is capable. However, this will not happen automatically. Although you have an excellent SCM system and a good tool and have completed the implementation successfully, you cannot assume that everything will go on nicely and without any problems.

The O&M phase of the SCM system is different from that of the software system(s). In the case of software, the operational phase commences when the system is turned over to the customer. Hence, the system is operational. The maintenance phase is implemented at the same time *if* the customer is paying the developing organization to maintain the system after delivery (i.e., for bug fixes, enhancements, and modifications). Otherwise, the system is considered "turn-key," and all activity ceases when the system is delivered (i.e., customers or buyers are on their own). In the case of the SCM system, the O&M phase starts once the system is implemented and the software development starts.

The operational and maintenance phase has to be carefully planned. During the O&M phase, the SCM sponsor should monitor the progress of the SCM system with the assistance of the project manager. The SCM sponsor or senior SCM manager is the most logical person to become the head of the SCM team—the CMO. He or she has experience, knowledge, and contacts with tool vendors and external consultants and has worked closely with the "SCM champions" in the organization. Accordingly, appointing this person as the CMO team is a very good idea as things will be easier for him or her than anyone else.

The main objective of the O&M phase is to ensure that the SCM system achieves its projected benefits—the users are satisfied, and there are no conflicts. The organization of the SCM system and SCM team will depend on the nature of the organization. As described in Chapter 14, there can be a central SCM team or there can be independent SCM teams for each project. In either case, the SCM team members report directly or indirectly to the CMO. The CMO maintains close relationships with the other support departments such as development, QA, testing, marketing, and technical support so that he or she can coordinate the SCM-related activities. For example, the marketing department has the major say in deciding the release date of a product, but there may be other factors that decide the release date like unfinished enhancements or unfinished changes, which only the development team is aware of. By coordinating and communicating with the representatives of all the various departments that have a role in the SCM activities, the CMO can eliminate interdepartmental conflicts and make sure that all departments are working toward a common goal.

The main activities of the CMO are ensuring that SCM activities are performed correctly, coordinating the SCM issues with the various departments, assessing the

deficiencies in the system and correcting them, assessing the training needs of existing and new users and giving them training, liaising with top management, giving progress reports, and obtaining resources.

Interdepartmental Coordination

As mentioned earlier, for the SCM system to function smoothly, the cooperation of all the departments is necessary. These departments might have conflicting interests. The marketing department might want the maximum number of features and the release of the system as early as possible. The development team might not be able incorporate all the enhancement requests and features and complete the development per the time schedule of the marketing team. The financial department might want to give additional resources to a project as there will be cost overruns. Managing these conflicting requirements is a very difficult task. The CMO can act as a liaison among these departments and between these departments and the company management and arrive at solutions that are satisfactory to all. For example, if the marketing team wants an early release of the product, then some of the features could be assigned to the next release.

SWOT Analysis

The CMO and the SCM team can assess the strengths, weaknesses, opportunities, and threats (SWOT) to the SCM system. The strengths should be reinforced so that they become ingrained in the organizational culture. The weaknesses should be rectified through appropriate corrective actions, so that the organization can be competitive. The opportunities and threats are identified at the strategic level by the management. However, these should be converted into operational level tasks and be performed by the various departments in the organization. For example, if the technical service department of the organization is finding it difficult to provide timely and quality service to the customers, it poses a threat to the organization as customers will become dissatisfied and might move to another vendor. If the SCM team can create a help desk of all the problems that have occurred in the past and how they were fixed, then the technical support team will be able to answer customer queries quickly (if the help desk has a previous instance of such a problem). If the help desk is kept up-to-date by adding the new problems and their fixes, over a period of time, the efficiency of the technical support team will increase dramatically. Similarly, many threats that affect the organization at the business level can be solved using SCM activities as SCM keeps track of all the activities, changes, problems and their solutions.

Documentation

During the implementation phase, the SCM plan is prepared, and the SCM system is designed. During the course of the implementation, the plan will be revised and updated to reflect the changes that occurred during implementation. The SCM plan is the fundamental document for performing SCM activities. It is the job of the CMO to ensure that all the SCM team members and the members of the organization have access to the latest copy of the SCM plan.

During the training phase, the vendors and the external consultants will prepare user manuals, procedures, and best practices documents for training and reference. These documents (their latest versions) should also be accessible to all users of the system. Such documents can be hosted on the company intranet or on the Internet (in the case of distributed teams). Access to these documents can be restricted or controlled using usernames and passwords.

Procedures and work instructions describe how tasks are carried out, the latter in more detail. Procedures and work instructions include process descriptions, roles and responsibilities, and process flow. The production of these documents is not a task for one person, but for those who define the processes. The process of producing this documentation commences during the SCM system design stage and develops through the implementation phase, at the end of which it is finalized. The procedures can be either in hard copy or in electronic form and can be posted on the company intranet or the company Web site. These procedures are actually part of the documentation control and should be revised following the appropriate change management procedures. Only the latest copy should be available for viewing.

Training

Training is a never-ending activity. One of the main tasks of the CMO or the SCM team is training new employees on the SCM concepts, tool usages, procedures, and best working practices. For this, the SCM team can conduct an induction program at regular intervals and ask the project managers to send their new team members for the training at the earliest possible occasion. Once the new employees are given the SCM training and told how SCM is practiced in the organization, they will do things the "right way" from the beginning.

Audits and Reviews

SCM tools allow most activities like check-in, check-out, and CR initiation to be performed by the developer. The SCM team should conduct periodic audits to ensure that these activities are performed correctly (i.e., check to ensure that people are not attempting to check things back into to the library without first going through the authorized review procedures). The SCM plan and the SCM system should also be reviewed and audited periodically.

The CMO should find qualified auditors and reviewers so that the CIs can be reviewed during the development stage and the FCAs and PCAs can be conducted before the system is handed over to the customer. The procedures and checklists for these reviews and audits should be developed and made available to the reviewers and auditors. The SCM database should be updated with the information regarding these reviews and audits.

CCB Formation

Another task of the CMO is the formation of CCBs. The number of CCBs in a company will vary depending on the size and complexity of the projects. Whether it is a single CCB or multiple CCBs or SCCBs or PCCBs, it is the job of the CMO

to find the qualified personnel for the CCB. Usually the CCB is chaired by the CMO or a senior member of the SCM team. Once the CR is initiated, the change management activities should be performed without any delay, so that the problem is resolved as soon as possible.

SCM Database Management

The CMO or the SCM team should also ensure that SCM database is up-to-date and not corrupted so that the status accounting function can be performed accurately. If the integrity of the data in the SCM database is not accurate, the information produced will be outdated, wrong, and useless. The status accounting function is the eyes and ears of the project and company management for tracking the progress of the project. So the SCM team should make it their priority to ensure that the right information is provided to the right person at the right time. This will improve the quality of the decisions and make the organization more effective and efficient.

Software Upgrades, Enhancements, and Modifications

The real value of a good SCM system is realized during the O&M phase. The SCM system has all the information about which files were changed, what the change was, why was it changed, what other files were changed along with it, when was it changed, what files went into a particular version of the system, what tools were used, and a lot of other similar details. In other words, SCM systems contain a recording of all the significant activities that happened during the development and evolution of the product or system. Chapter 8, outlines the change and problem management process and the creation of help desks. The SCM database together with the help desks provides an environment where software could be upgraded, bugs could be identified and fixed, and enhancements could be incorporated. All these tasks could be performed quickly and without any wasted effort since we know exactly where to look for them and which items need modification.

Most software companies support more than one version of their software. For example, when the latest version of a product is 8.0, there may be many users still using the older versions, even version 1.0. So if some user from Timbuktu, who is using version 1.0, encounters a problem and calls technical support, the support team might have to recreate the version 1.0 and then find out the source of the problem. This is possible, thanks to SCM; one just has to run the build script that created version 1.0. (with the appropriate software and compiler versions). Since the build scripts are under configuration control, they will be readily available in the SCM library. Thus the bug can be fixed, the bug-fixed version can be delivered to the customer, and the organization will have one very satisfied customer in Timbuktu. This scenario comes with a caveat. An earlier version of the software might also have been running on different hardware. SCM by itself cannot handle this—unless perhaps there is some virtual machine simulation. In general, however, this is a fundamental and natural limitation of SCM. Ultimately, the hardware can restrict what might be runnable on it. A company might, for this reason, make an explicit policy about what earlier versions of its software that it will continue to support.

The capability to deliver bug-fixed versions faster to the customers is very important in this age where awareness of bugs in the software spreads very fast (through the Internet). So if a flaw is identified, the time it takes the company to release a bug-fixed version to the customer is very crucial. If the flaw is a security problem that leaves the software vulnerable to attacks from hackers or viruses, then the gravity of the situation increases exponentially. If SCM practices were not around to help organizations in these areas, then many companies that release patches almost on a daily basis would have filed for Chapter 11 bankruptcy protection by now!

Help Desks

Help desks are repositories of organizational knowledge that can be used by the maintenance and support team. The help desk contains information on many topics including correct operating practices, problems with the software, how to solve these problems, details of CRs or PRs, and resolution. When encountered with PRs from customers, the maintenance team and support personnel can query the help desk to see if a problem that is similar to or the same as the current problem has occurred in the past. If there are past occurrences of similar or same problems, then problem resolution will be quick. To get the maximum benefit out of the help desks, help desks should be designed properly. They should contain all the details arranged and stored so that it can be retrieved easily and intuitively. Many companies post FAQs and their answers on their support Web sites and encourage users to first check the FAQ before calling technical support. A well-organized and -categorized FAQ can substantially reduce the workload of technical support personnel. As new problems and their solutions get added to the help desk, they will simultaneously be updated on the Web site.

Change and Problem Requests from Customers and In-Field Emergency Fixes

Change and problem requests can come not only from the QA and testing team but also from the users once the system is released for general use. These requests (both internal and external) are handled as described in Chapter 8, by following formal change management and control procedures. Emergency fixes are required when a problem occurs, and there is not enough time to resolve it using formal change management and control procedures. Here the priority is to find and fix the bug and resolve the issue as soon as possible so that the customer can continue using the product. Emergency fixes are not left unattended after the emergency. Once the emergency is over, when the organization has enough resources, procedures such as change evaluation, impact analysis, and CCB meetings are conducted. The proper documentation is created, and the SCM database is updated with these details. This is to ensure that the integrity of the data is maintained. If the emergency fix is left once the fix is delivered to the customer, then there will not be any record of it in the SCM database; this can result in various serious problems that the SCM is trying to prevent.

Reusability Improvement

Every project will have reusable items. Reusable items are CIs that can be used in more than one place. When the number of reused CIs increases, the number of items to be developed decreases. If the same item can be used by different subsystems of the system, then there is no point in the two groups developing two different versions of a program or function that performs the same thing. With increased reuse, the "reinventing the wheel" phenomenon can be avoided. Thus, increased reusability will have a direct impact on productivity as it saves design, development, and testing time. Also, it will reduce the number of CIs that need to be managed, thereby reducing the SCM workload. To promote reusability, the designers and developers of the project should know that, there exists a CI that performs the same function as the one they are designing or developing. So the SCM team should identify CIs that have the potential to be reused and place them in a separate controlled library usually known as the reusability library. The items in the reusability library are also under SCM control, but they are placed in separate library to promote reusability. An index of the CIs and a small description of the function(s) performed by each CI can help programmers and developers in searching for and identifying the components that they can use, before starting to develop the components once again. Taking this one step further, there can be a reusability library for the entire organization that can be under the control of the central SCM team, where the reusable items from the different projects are placed and managed. This will improve the reusability at the organizational level. In many cases, many items developed, tested, reviewed, and approved for one project could very well be used in another project. Date manipulation functions, string manipulation functions, and programs for displaying error messages are some of the items that have uses in almost all projects.

Metrics

You have implemented and are operating the SCM system, but how will you know that the SCM system is delivering the promised improvements? More importantly, how will you know that the SCM system is helping in improving your software development process? You should measure the various parameters of the system. If you do not measure your current performance and use the data to improve your future work estimates, those estimates will just be guesses. Because today's current data becomes tomorrow's historical data, it is never too late to start recording key information about your project. You cannot track project status meaningfully unless you know the actual effort and time spent on each task compared to your plans. You cannot sensibly decide whether your product is stable enough to ship unless you are tracking the rates at which your team is finding and fixing defects. You cannot quantify how well your new development processes are working without some measure of your current performance and a baseline to compare against. Metrics help you better control your software projects and learn more about the way your SCM system and organization works. As time goes on and people become more comfortable with the system, performance should improve. You will able to measure improvement only if you have measured the various parameters from the

beginning. If the metrics reveal that the performance is not improving, then management should investigate the reasons and take the necessary corrective actions. The following are some of the SCM-related parameters that you can measure.

- Average time taken for the resolution of a CR;
- Average time taken to resolve a technical support query;
- Number of CRs and PRs;
- Percentage of approved CRs and PRs;
- Percentage of time spent on development;
- Percentage of time spent on tasks such as testing, QA, and debugging;
- Number of defects found after each release;
- Number and type of changes (e.g., bugs, enhancements, and emergency fixes);
- Number and severity of defects found during development;
- Number and severity of defects found during testing;
- Average time taken to identify the source of a defect (defect identification);
- Average time taken to resolve a defect;
- Amount of reused code;
- Number of unfixed bugs in each release;
- Number of enhancements in each release;
- Average number of CIs impacted by a CR or PR;
- Product development cycle time;
- Difference between the estimated and actual values for each activity.

The above list is by no means an exhaustive one. It is just a representative list of what can measured. You should decide what more should be measured depending on the nature of your organization and projects.

SCM Maintenance Phase

To function properly, the SCM system needs regular maintenance. The SCM plan needs revision and updating to adapt to the changing situations in the organization. We have already determined that the SCM system should be reviewed regularly, with the review comments and suggestions incorporated into the system. Also, the SCM system needs fine-tuning as employees become familiar with it. Once the SCM system has reached a stable state, necessary actions should be taken to improve the performance. The SCM metrics discussed in the previous section can indicate whether the SCM system is functioning properly or not.

The SCM tools that are implemented are another area that needs maintenance. The CMO should be in regular contact with the vendors to see whether any upgrades or updates are available. All patches and upgrades should be installed to ensure that the tools are working at their maximum efficiency. Employees should be given refresher courses on the new functionality that gets added with each new upgrade. The training documentation should also be updated so that it is in sync with the procedures and processes.

Summary

Proper O&M of an SCM system after it has been successfully implemented requires careful planning on the part of the SCM team and adherence to the best practices by the users of the system. Moreover, the support and sponsorship of top management is absolutely essential to the successful O&M of the SCM system.

Selected Bibliography

Buckley, F. J., "Implementing a Software Configuration Management Environment," *IEEE Computer*, July 1994, pp. 56–61.

CM Crossroads: The Configuration Management Community (http://www.cmcrossroads.com/).

Dart, S., "Achieving the Best Possible Configuration Management Solution," *Crosstalk: The Journal of Defense Software Engineering*, September 1996.

Dart, S., "To Change or Not to Change," *Application Development Trends*, Vo1.4, No. 6, 1997, pp. 55–57.

Dart, S., *Configuration Management: The Missing Link in Web Engineering*, Norwood, MA: Artech House, 2000.

Estublier, J., "Software Configuration Management: A Roadmap," *Proceedings of the Conference on the Future of Software Engineering*, Limerick, Ireland, 2000, pp.279–289.

Feiler, P. H., "Software Configuration Management: Advances in Software Development Environments," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1990.

Irish, D. E., "Putting the Horse Before the Cart: Preparing Your Staff for Project Management Software," *Proc. ACM SIGUCCS 2001*, Association of Computing Machinery, 2001, pp. 59–62.

Kolvik, S., "Introducing Configuration Management in an Organization," *Proc. ICSE '96 SCM-6 Workshops (Selected Papers)*, Berlin, Springer-Verlag, 1996, pp. 220–230.

Moor, S. R., J. Gunne-Braden, and K. J. Gleen, "Enterprise Configuration Management—Controlling Integration Complexity," *BT Technology Journal*, Vol. 15, No. 3, July 1997, pp. 61–72.

Tellioğlu, H., and I. Wagner, "Negotiating Boundaries: Configuration Management in Software Development Teams," *Computer Supported Cooperative Work (CSCW): The Journal of Collaborative Computing*, Vol. 6, No. 4, 1997, pp. 251–274.

Thompson, S. M., "Configuration Management—Keeping it all Together," *BT Technology Journal*, Vol. 15, No. 3, July 1997, pp. 48–60.

CHAPTER 23

SCM in Special Circumstances

Introduction

SCM can be practiced in a variety of situations. Projects where SCM is practiced can vary from small, single-person projects to very large and complex projects involving hundreds of people. Even though SCM concepts are the same irrespective of the size of the project, factors such as the way in which SCM is practiced, the procedures followed, the degree of control and presence of formal procedures, the use of SCM tools, and the level of automation are not the same. They will vary from project to project.

Also, development environments are changing. Now we have integrated development environments (IDEs) and CASE environments. These are quite different from earlier project environments. Today a software system can be cross-platform (the software system can span more than one hardware or software platform) and can involve more than one development environment. The use of CASE tools for application development is now commonplace. Moreover, today, there are distributed development environments with development happening in different parts of the world. This chapter briefly discusses how CM is practiced in these situations.

SCM and Project Size

The size and number of people involved in a project can definitely have an impact on how SCM is practiced in a project. There can be, for example, single-person projects, a single person managing more than one project, projects involving more than one person, and projects involving thousands of people. In the case of a single person doing a project, SCM is not an absolute must, because issues such as communications breakdowns, shared data problems, and simultaneous update problems are not encountered. Even in this kind of a project, however, practicing SCM is useful because due to human characteristics such as carelessness, oversight, and forgetfulness, work that has already been done can get overwritten or the same problem might be solved more than once, among other possible mishaps.

In addition, the person who is doing the project will not be around forever. So when a new person takes charge of the project, he or she has to be able to access the necessary information. So if there is no documentation and no records—and the only record of what happened to the project is in the other person's head—then it will be difficult for the new person to manage the project. Questions such as why a change was made, what items were changed, and if a certain module is changed

which items will be affected will not have answers. To get these answers, the new person will have to go through all of the programs and hope to get lucky.

So even in the case of small projects where only one person is involved, practicing SCM is a good idea. Here, there is no need to use an SCM tool and formal change management procedures with CCB meetings and the like. An informal SCM system contains information on topics such as changes that have occurred, the reasons for changes, the item dependencies, and versions and releases are documented. That would be more than enough. If the organization has an SCM tool and if all the developers have been trained on the tool, then even small projects can benefit from using an SCM tool.

In the case of larger projects involving many people, SCM must be practiced. Earlier chapters discuss why practicing SCM is important. It is always advantageous to automate the various SCM functions using SCM tools, because it will improve development productivity and reduce errors. In today's brutally competitive world, where everyone is fighting for survival and market share, SCM can be used as a strategic weapon that will give the organization an edge over others that are not using SCM or that are using it less effectively.

SCM in Very Large Projects

The SCM system and management of the SCM system is different in the case of very large projects. By very large software projects, I mean a project that has more than 100 team members at a time, a project that requires the effort of more than 1,000 human years, a project where a lot of work is subcontracted and there are a large number of subcontractors, a project that involves the development of an application for many platforms, a project that is cross-platform, a project that is developed by more than one geographically distributed team, or a project that is a combination of any of the above criteria. These are some indicative figures and criteria, and to qualify as a very large project, the project has to involve a lot of people and must have millions of lines of code.

Here, the emphasis is not on the complexity or the criticality of the project, but on the size of the project and project team, the number of groups involved, and other similar factors. The SCM principles used in these projects are the same as those used in small projects or in mission-critical projects, but what makes these projects different is the sheer size and the management and organizational challenges that such a size poses. Some examples of such systems include database management systems (a system like Oracle or DB2), ERP systems (something like SAP R/3 or the PeopleSoft ERP system), and operating systems (Windows, UNIX, MVS). Typically, these projects will have many modules or subsystems, each of which can function in a semiautonomous fashion. These independent subsystems may be developed by different teams from different companies in different geographical locations.

Management of such complex systems is impossible without a good SCM system. This is because in projects of such large scale, the chances of all of the classical problems—communications breakdown, the shared data problem, the simultaneous update problem, and the multiple maintenance problem—occurring are very

high. CM in these projects is very formal in nature with numerous controls and procedures. It is practically impossible to do the CM of these projects manually. So these projects ideally should use a high-end SCM tool.

Performance of SCM Tools

The SCM tool used in a very large project should be capable of supporting hundreds of developers and testers with tens of millions of lines of code. The system should do this without degradation of performance. Accordingly, the tool selection process must take these performance factors into account. Questions such as how many people will be using the system concurrently, how much data will have to be handled, and does the system have to be up around the clock (to support people in different time zones should be considered during selection, and the implication of these factors on the performance of the tool must be analyzed. Only that tool that can take the load without compromising performance should be selected.

Implementation Strategy

Chapter 15 explains that large projects use SCM tools that fall into a category called process-oriented tools. According to Dart [1], these tools include version control capabilities and at least some of developer-oriented capabilities. These tools have the ability to automate software flow life cycles, roles, and their responsibilities and to customize the out-of-the-box process model. These tools provide an integrated approach to change management where problem tracking is associated with the code.

So in a very large project where formal procedures are to be implemented and automated, and tools have to carry out more than just version control, these types of tools will be used. Here the key phrases are process dependency and information automation and integration. A very large project by its nature requires formalized and automated procedures, so a tool that is capable of taking care of these things is an absolute necessity.

During the implementation of the tool, as discussed earlier, it is better to start with a pilot project. In this case, the pilot project will be a module or subsystem of the project. Because these kinds of projects will be using a lot of automated tools like CASE tools, code generators, test data generators, automatic testing tools, and code analyzers, how the SCM tool will integrate and interact with these tools must be analyzed and studied. If possible, it is better to automate the information flow from the other tools used in the project to the SCM tool repository, so that there is no wasted effort. Suitable interfaces have to be built or bought. In some cases, however, it might be wise to enter the information manually from the tools used in the project into the SCM tool repository, because making or buying an interfacing system could be very costly. So here project management will have to do a cost-benefit analysis and decide which strategy to adopt.

During pilot project implementation, all of these issues should be addressed, and solutions should be identified. It is better to choose a module or subsystem that is representative and contains all potential elements (tools and other complexities) for the pilot project and face the problems head-on, because doing so will provide

much data on how to implement the SCM system and SCM tool in other modules or subsystems. Accordingly, in the case of very large projects, you should choose the module that is most difficult and most complex as the pilot because of all of the difficulties that could be encountered and solved during the pilot project phase itself.

In addition, during the pilot project phase, because the SCM implementation team, the SCM experts, and the tool vendor's representatives will be concentrating on one project, problems may be solved more effectively and efficiently. In very large projects, it is better to have the SCM system in place from the initial phases onward. So if the pilot implementation can be done in a simulated environment where all the tools and other elements representative of the project in question are present, this will give developers the opportunity to understand the problems, solve them, and then to implement the SCM system and the tools, in the project from the initial phase itself. However, the simulated environment that is being created for the pilot implementation should be a representative model of the actual project environment.

Distributed, Concurrent, and Parallel Development

Very large projects are characterized by their distributed nature where concurrent and parallel development is commonplace. At any given time, many people will be working on the same programs or different variants of it. So the SCM tool that is used in a very large project should be very good at managing variants (branches that will not be merged) and temporary branches (branches that will be merged).

The merging capabilities of these tools also have to be very good. Any tool that is used in a very large project should be capable of supporting distributed development. Now, development teams work from different continents, creating different subsystems or modules of a software product. To treat these physically distributed systems as a single logical entity and to manage them in such a fashion that the project managers are not bothered with the underlying complexities is a must for a tool that is used in a very large project.

Change Management

In large projects, the change management activities cannot be handled by a single change control authority. Accordingly, there will be multiple CCBs and in many cases multilevel CCBs. Change initiation, change request routing, and change disposition can be automated using SCM tools. However, people are needed to make decisions, so CCB members should be trained to use the technology.

Because there are many CCBs of equal status and priority, there should be a super CCB (SCCB) that is authorized to resolve conflicts between the multiple CCBs. The guidelines for the functioning of the CCBs and how to resolve conflicts should be well-documented, and if required, they can be automated (a rule-based system) so that the SCM tool can take appropriate actions without human intervention. For example, you need to consider what to do if there is a tie among the members. Should the problem be escalated to the higher level CCB, or should the members be informed about the poll result and asked to vote again or to attend a physical CCB meeting? These rules can be coded into the system and many procedures automated,

thus making the best and most effective use of the CCB member's time—and reducing the time in the disposition of CRs.

Status Accounting

Status accounting is the function that documents and reports the information related to CIs to everyone involved in the project. The status accounting function should also be able to answer ad hoc queries. In a large project, creating and distributing reports, even the routine reports, as hardcopy is expensive—not to mention that it is an administrative nightmare. So in this kind of a project it is better to publish and post these reports on the corporate intranet and inform staff about the fact that the reports are available via e-mail or any other messaging system that the project is using.

For ad hoc querying, it is better to assign roles—such as developer, tester, and manager—and give selective access to the people who fit these roles. For example, a developer could be given access to query the table that contains information about the CIs of his or her module, whereas a manager could be given the access to query any table in the database. This kind of electronic distribution of the status accounting information is necessary for a large project, because the other option of having hardcopies is a big burden on the SCM budget.

System Building

In the case of very large products, system building can take many hours or even days. Accordingly, the frequency of the builds is important because much time and money is involved with each build. This points to the need for a build strategy.

There are two types of builds: clean and nonclean. A clean build is the process of starting with only the source items and then building the entire system step-by-step from those source items. A nonclean build uses some derived items as inputs for the build process. For example, in the case of a nonclean build, all subsystems that were not changed could be used as is for system building, whereas in the case of a clean build even the subsystems or modules that were not changed would be built again from the source components. In the case of very large projects, having a clean build each and every time is neither practical nor needed. The system could be built using derived components or subsystems that were not changed. This will save a lot of time, especially during the integration testing and alpha and beta testing phases.

For the final release, a clean build is best. As discussed earlier, we will use SCM tools in such a large project. The build capabilities of these tools should be adequate enough to produce accurate and reliable builds.

Skill Inventory Database

The skill inventory database, detailed in Chapter 15, is an absolute must in a large project. The two main reasons are that (1) there will be a huge demand for people to do tasks such as change evaluation, impact analysis, and auditing, and (2) in a large project, the skills and availability of all the people involved with the project

are difficult to remember and should therefore be recorded somewhere. So, the best, fastest, and easiest method for finding the right people to get the job done is to store the details about the people in a database.

Training

In a very large project, the SCM training of the team members is a very important issue. Because these projects span many years, many different people will be involved with the project. So once the team member training done during the implementation phase is over, a system should be in place to train new members who join the project at a later date.

All large projects have induction programs, which the new members have to undertake. These programs usually give a general idea about the project, the major components, and the different functions that is sufficient to give users a bird's-eye view of the project. This is important, because only if one knows the big picture can one understand the consequences of his or her actions in a module or subsystem. The SCM and SCM tool training should be a part of the induction program, so that all the new members will be trained in that also.

Help Desks and Other Knowledge-Sharing Systems

As discussed previously, the team in a very large project is transient in nature. Hence, it is important for all events in a project to be recorded, including how a problem was identified, how it was fixed, how a bug escaped the testing phase, and what points developers should be aware of when using a tool. This information should be captured in a knowledge base or help desk and made available to the team members.

This knowledge-capturing and -sharing function is very important in a very large project, because the chances of problems recurring and people reinventing the wheel are greater. If all experiences are documented (maybe using an expert system), much time can be saved and development productivity improved. Also, these help desks are invaluable to the technical support teams and system maintenance personnel.

SCM Costs

It is difficult to estimate the cost, effort, time, or size of very large systems with a high degree of accuracy. Even in small- and medium-sized projects, the hidden costs of SCM implementation, discussed in Chapter 18, apply. In large projects SCM implementation costs will be even more difficult to predict because many factors are difficult to estimate due to project duration. Estimating what the scenario will be five years down the road is quite difficult.

Being able to foresee the future with unerring accuracy is not a task that is easily accomplished by people or by machines. Also, the real world does not stand still while large systems are developed; new products and processes are discovered, underlying assumptions are invalidated, new laws are passed, and developers learn new things. So any estimates about SCM implementation and postimplementation

will have to be reevaluated frequently in the case of very large projects, so that the estimates and budgets can be updated.

Concurrent and Parallel Development

The days of one-at-a-time modifications to CIs are a thing of the past. Today's SCM tools can support parallel branches for concurrent development and variant development. In the case of parallel branches that are used for concurrent development, the sophisticated tools allow more than one user to make modifications to the same file(s), and then the tool merges those parallel versions. The merging capabilities of the SCM tools are increasing, and that makes the life of the person doing the merging a lot easier. Now the tools can compare the different parallel versions against a common ancestor and highlight the areas where they are different and where the changes have been made. According to Burrows [2], the capability of modern merge tools is now so strong that users are tempted to accept the tools' automatic resolutions and omit essential testing processes, which is not at all recommended.

Web Site Management

Web design and development is a totally different ballgame from the normal software development process. Web sites, to attract visitors and encourage them to come back, have to change their content and offerings very frequently. An advantage of having a Web site is providing up-to-the-minute information to users. Accordingly, managing the changes to a Web site is more difficult than that of a software product, because of the rate at which the contents of the Web site changes. It is not only the contents that change; to make the pages attractive and catchy, elements such as the presentation styles, the design elements, and the layout are changed very often.

Another factor that makes Web development different from software development is the number of CIs that must be managed. Even a medium-sized company's Web site will have more than 1,000 pages, each containing various objects and elements such as downloads, pictures, and movie clips—so we are talking about thousands of objects. Even though some of the modern full-fledged SCM tools can handle a huge number of CIs, traditional SCM tools were never designed to manage this huge number of CIs and, even if they do, it will be at the cost of performance.

Another capability that is required for the management of the Internet sites is the ability to recreate a Web page as it was on a particular day or time. The rate at which the contents change is extremely rapid (in many cases almost on a daily basis)—in fact, in some cases (sites pertaining to, e.g., weather, stocks, and airlines), the content changes more often than once a minute. As a result, the information required for the build management tools is phenomenal, and ensuring repeatable and reliable rebuilds of the thousands of versions of the Web site is quite a challenge. Consequently, the CM of Web sites and Internet sites requires different skills than those used for software management. According to Burrows [2], CM support for Web and particularly intranet pages and their embedded objects is creating an

important new market for the vendors of CM tools that, in time, could exceed the size of the market for managing software development.

SCM in Integrated Development Environments

An integrated development environment (IDE) is a programming environment integrated into an application. So an IDE is a set of programs that runs from a single user interface. Some of the most popular IDEs include Visual C++, PowerBuilder, and Visual Basic.

The advantage of using an IDE is that you can design, develop, test, debug, and run applications without leaving the development environment. So when using an IDE, it is quite natural to assume that even for SCM functions one does not have to leave the development environment. Today's SCM tools do integrate with IDEs such as Visual C++, Visual Basic, and PowerBuilder so that developers need not go outside the IDE for SCM functions. The same is true with CASE tools in which the SCM tools share information from the CASE repository. Many of today's SCM tools integrate seamlessly with the integrated environments, so that they become part of the environment. Therefore, when carrying out SCM functions, developers do not have to leave the development environment. Some examples of tools are Continuus CM integrating with Visual C++, PVCS integrating with PowerBuilder, and Visual SourceSafe integrating with Microsoft's IDEs. Thus, these tools make the CM process in an IDE more intuitive and painless.

In the future, we will see seamless integration (integration without interfaces and interface packages) of the SCM tools, and IDEs and CASE tools will incorporate SCM functionality. Thus the development environment will become truly integrated. In fact, the day is not far off when SCM tools will supply information or merge with project management tools, thus enabling seamless information integration and easier and more efficient project management.

SCM in Distributed Environments

Today, the development of a software system is done by many teams distributed across different parts of the globe. Different teams working from different locations develop a software product or system. This is an ideal situation for SCM because lack of control can lead to chaos and result in project failures. With advancements in telecommunications and networking technologies, distributed computing is becoming easier and easier. The capability to manage distributed development is now being offered by many tools. As communications and information technology make rapid strides forward, distributed development is going to be commonplace, and the distributed development capabilities of SCM tools are going to get better and better.

Today, project teams that are thousands of miles apart can work as if they share the same office. SCM tools have evolved to incorporate these technologies. Today's SCM tools are capable of supporting distributed development, parallel development, and concurrent development. The capabilities and features of modern SCM tools are so advanced that users do not have to bother about the complexities involved.

Such systems will manage issues such as networking, communication, security, and concurrency management.

SCM and Case Tools

CASE tools provide automated methods for designing and documenting traditional structured programming techniques. The ultimate goal of CASE is to provide a language for describing the overall system that is sufficient to generate all the necessary programs. Thus, a CASE tool is a software package that is used for developing an information system. It is used in all phases of software development: analysis, design, programming, and testing. For example, data dictionaries and programming tools aid in the analysis and design phase, while application generators help speed the programming phase. Automated testing tools and test data generators help in the testing phase.

Most CASE tools store project information in their repositories, and they use this information for processes such as code generation, test plan creation, and test data generation. Because a CASE tool uses the information from its repository to generate application codes, the information stored in a CASE tool's repository is very detailed. For example, many CASE tools use the entity-relation (ER) models as a starting point for analysis. So their repositories will contain details including the various entities and their attributes. These repositories also contain information about the interdependencies of the various objects and programs, because this type of information is required for application generation and testing.

We have seen that SCM systems also use the same information as CASE tools (although maybe not to the same level of detail) to function. So a CASE tool's repository contains the information that is needed for an SCM system. Many CASE tools have rudimentary SCM functions built into them. For example, CASE tools can manage changes and keep an audit trail of modifications to the changes to an item.

CASE tools do not have the full functionality of SCM tools, but they do provide the inputs that are required by the SCM system. Thus, if CASE tools and SCM systems can be integrated, so that the SCM systems draw the information from the CASE tools and then perform the SCM functions, much time and effort that would otherwise be spent on information gathering can be saved. The ideal solution to this scenario is that of a CASE tool that is so tightly integrated with an SCM tool that they share the same repository. Hence, users could perform all SCM functions without leaving the CASE environment.

Summary

This chapter examines some of the special situations where SCM is practiced. The basic SCM concepts are the same irrespective of the situation in which it is used. Only the application of these concepts varies. Factors such as the degree of formalism, the presence or absence of certain procedures, and the structure of the SCM organization will vary depending on the situation. As technology takes rapid strides and new and more efficient methods of developing software emerge and become

popular, the way in which SCM is practiced is also changing. SCM tools are also evolving to support these new development paradigms and methodologies.

References

[1] Dart, S., "Not All Tools Are Created Equal," *Application Development Trends*, Vol. 3, No. 9, 1996, pp. 45–48.

[2] Burrows, C., "Configuration Management: Coming of Age in the Year 2000," *Crosstalk: The Journal of Defense Software Engineering*, Mar. 1999, pp. 12–16.

Selected Bibliography

Babich, W. A., *Software Configuration Management: Coordination for Team Productivity*, Boston, MA: Addison Wesley, 1986.

Ben-Menachem, M., *Software Configuration Guidebook*, London: McGraw-Hill International, 1994.

Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, 1992.

Bersoff, E. H., V. D. Henderson, and S. G. Siegel, *Software Configuration Management: An Investment in Product Integrity*, Englewood Cliffs, NJ: Prentice-Hall, 1980.

Burrows, C., and I. Wesley, *Ovum Evaluates: Configuration Management*, London: Ovum Limited, 1998.

Conradi, R. (ed.), *Software Configuration Management: ICSE'97 SCM-7 Workshop Proc.*, Berlin: Springer-Verlag, 1997.

Denning, D. E., *Information Warfare and Security*, Reading, MA: Addison-Wesley Longman, 1999.

Heiman, R. V., S. Garone, and S. D. Hendrick, "Development Life-Cycle Management: 1999 Worldwide Markets and Trends," Technical Report, Framingham, MA: International Data Corporation, June 1999.

Magnusson, B. (ed.), *System Configuration Management: ECOOP'98 SCM-8 Symp. Proc.*, Berlin: Springer-Verlag, 1998.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, 1991.

APPENDIX A

SCM Resources on the Internet

Organizations and Institutes

American National Standards Institute (ANSI) (http://www.ansi.org)

Association for Computing Machinery (http://www.acm.org)

Association for Configuration and Data Management (ACDM) (http://www.acdm.org/)

Configuration Management Specialist Group of British Computer Society (CMSG) Home Page (http://www.bcs-cmsg.org.uk/)

Configuration Management, Inc. (CMI) (http://www.cmi.com/)

Institute of Configuration Management—The home of CM II (http://www.icmhq.com/index.html)

Institute of Electrical and Electronics Engineers, Inc. (IEEE) (http://www.ieee.org)

International Organization for Standardization (ISO) (http://www.iso.ch/)

International Society for Configuration Management (ISCM) (http://www.cmtf.com/society.html)

National Aeronautics & Space Administration (NASA) (http://www.nasa.gov)

Software Engineering Institute (http://www.sei.cmu.edu/l)

Software Technology Support Center (STSC) Home Page (http://www.stsc.hill.af.mil/home.asp)

Resource Pages

Bob Aiello's CM Best Practices Website (http://cmbestpractices.com/)

CM Crossroads: The Configuration Management Community (http://www.cmcrossroads.com/)

SCM Links by Mária Bieliková (http://www2.fiit.stuba.sk/~bielik/scm/links.html)

Steve Easterbrook's CM Resource Guide On-Line (https://www.cmpic.com/cmresourceguide.htm)

The Configuration Management Process Improvement Center (https://www.cmpic.com/index.html)

Commercial Research Organizations

Forrester Research (http://www.forrester.com/)

International Data Corporation (IDC) (http://www.idc.com)

Ovum Ltd. (http://www.ovum.com)

Digital/On-Line Libraries

ACM Digital Library (http://dl.acm.org/)

IEEE Computer Society Digital Library (http://www.computer.org/csdl)

IEEE Standards Association (http://standards.ieee.org/)

IEEE Xplore Digital Library (http://ieeexplore.ieee.org/Xplore/home.jsp)

Magazines and Periodicals

ACM Transactions on Software Engineering and Methodology (http://tosem.acm.org/)

Application Development Trends (http://www.adtmag.com/)

Communications of the ACM (http://cacm.acm.org/)

Computing in Science & Engineering (http://www.computer.org/portal/web/computingnow/cise)

Computing Reviews (http://www.computingreviews.com/)

Cross Talk Home (http://www.crosstalkonline.org/)

Cross Talk Magazine (http://www.crosstalkonline.org/)

IEEE Annals of the History of Computing (http://www.computer.org/portal/web/computingnow/annals)

IEEE Computer (http://www.computer.org/portal/web/computingnow/computer)

IEEE IT Professional (http://www.computer.org/portal/web/computingnow/itpro)

IEEE Software (http://www.computer.org/software/)

IEEE Spectrum (http://www.spectrum.ieee.org/)

IEEE Transactions on Computers (http://www.computer.org/portal/web/tc)

IEEE Transactions on Software Engineering (http://www.computer.org/portal/web/tse)

Journal of the ACM (http://jacm.acm.org/)

General Sites

comp.software.config-mgmt FAQ: Configuration Management Tools Summary (http://www.faqs.org/faqs/sw-config-mgmt/cm-tools/)

comp.software.config-mgmt FAQ: General Questions (http://www.faqs.org/faqs/sw-config-mgmt/faq/)

comp.software.config-mgmt FAQ: Problem Management Tools Summary (http://www.faqs.org/faqs/sw-config-mgmt/prob-mgt-tools/)

Department of Defense Single Stock Point (DODSSP) for Military Specifications, Standards and Related Publications (http://dodssp.daps.dla.mil/)

Hal Render's Bibliography on Software Configuration Management (http://liinwww.ira.uka.de/bibliography/SE/scm.html)

Ken Rigby's Configuration Management Glossary (http://www.personal.kent.edu/~plucasst/NASAGRC/glennglossary.htm)

Ken Rigby's Configuration Management Plan—Model Text (http://ncsx.pppl.gov/SystemsEngineering/Plans_Procedures/NCSX_Mgmt_Plans/CMP/Archives/CONFIGURATION%20MANAGEMENT%20PLAN%20-%20Model%20text.htm)

List of the 20 most popular CM standards by Software Engineering Process Technology (SEPT) (http://www.12207.com/test.htm)

Military Standards MIL-STD, Military Specification MIL-SPEC (http://www.everyspec.com/)

NASA Software Configuration Management Guidebook (http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19980228473.pdf)

NASA's Software Assurance Technology Center (SATC) (http://satc.gsfc.nasa.gov/)

Software Configuration Management Index (http://www.faqs.org/faqs/sw-config-mgmt/)

Major SCM Tools

AccuRev/CM (http://www.accurev.com/)

AllChange (http://www.intasoft.net/default.asp)

assyst (http://www.axiossystems.com/en/solutions/itsm/assyst-itsm.html)

CA Harvest Software Change Manager (http://www.ca.com/us/devcenter/ca-harvest-software-change-manager.aspx)

ChangeMan SSM(http://www.serena.com/index.php/en/products/mainframe/changeman-ssm/)

ChangeMan ZMF (http://www.serena.com/index.php/en/products/mainframe/changeman-zmf/)

ClearCase (http://www-03.ibm.com/software/products/en/clearcase)

Dimensions CM (http://www.serena.com/index.php/en/products/featured-products/dimensions-cm/)

ISPW (http://www.ispw.com/)

PTC Integrity (http://www.mks.com/platform/our-product)

Rational Synergy (http://www-03.ibm.com/software/products/en/ratisyne)

Razor (http://www.visible.com/Products/Razor/)

Spectrum SCM (http://www.spectrumscm.com/)

StartTeam (http://www.borland.com/products/starteam/)

TRUEChange (http://www.mccabe.com/cm.htm)

Note: The preceding list contains only the popular high-end SCM tools. For a comprehensive list and comparison of the tools, users should check the following sites, which are regularly updated:

- http://en.wikipedia.org/wiki/List_of_revision_control_software
- http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- http://en.wikipedia.org/wiki/Comparison_of_open_source_configuration_management_software

APPENDIX B

SCM Bibliography

Abu-Shakra, M., and G. L. Fisher, "Multi-grain Version Control in the Historian System," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 46–56.

ACM Staff (ed.), *Proceedings of the 3rd International Workshop on Software Configuration Management*, New York: Association for Computing Machinery, 1991.

ACM Staff (ed.), *Second International Workshop: Software Configuration Management Proceedings*, New York: Association for Computing Machinery, 1989.

Adams, C., "Why Can't I Buy an SCM Tool?" Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 278–281.

Adams, P., and M. Solomon, "An Overview of the CAPITAL Software Development Environment," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 1–34.

Adams, R. J., and S. Eslinger, "Lessons Learned From Using COTS Software on Space Systems," *Crosstalk: The Journal of Defense Software Engineering*, June 2001, pp. 25–30.

Aiello, B., and L. Sachs, Software *Configuration Management Best Practices*, Upper Saddle River, NJ: Addison-Wesley, 2011.

Alain Abran, A., and J. W. Moore (eds.), *SWEBOK: Guide to the Software Engineering Body of Knowledge (Trial Version)*, Los Alamitos, California: IEEE Computer Society, 2001.

Alder, P. S., and A. Shenhar, "Adapting Your Technological Base: The Organizational Challenge," *Sloan Management Review*, Fall 1990, pp. 25–37.

Alder, R. S., "Today's Software Complexity Demands Good CM," *Crosstalk: The Journal of Defense Software Engineering*, February 1998, p. 2.

Allen, L., et al., "ClearClase MulitSite: Supporting Geographically Distributed Software Development," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 194–214.

Ambriola, V., and L. Bendix, "Object-Oriented Configuration Control," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 135–136.

Andriole, S. J., *Managing Systems Requirements: Methods, Tools, and Cases*, New York: McGraw-Hill Companies, Inc., 1996.

Angstadt, B.L., "SCM: More than Support and Control," *Crosstalk: The Journal of Defense Software Engineering*, March 2000, pp. 26–27.

ANSI/IEEE Std-1028-1988-Standard for Software Reviews and Audits, 1988.

ANSI/IEEE Std-1042-1987–IEEE Guide to Software Configuration Management, 1987.

ANSI/IEEE Std-730.1-1995–IEEE Guide for Software Quality Assurance Planning, 1995

ANSI/IEEE Std-730-1998–IEEE Standard for Software Quality Assurance Plans, 1998.

Aquilino, D., et al., "Supporting Reuse and Configuration: A Port Based SCM Model," *Proceedings of the 3rd International Workshop on Software Configuration Management*,

Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 62–67.

Asklund, U., and B. Magnusson, "A Case-study of Configuration Management with ClearCase in an Industrial Environment," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 201–221.

Atkins, D. L., "Version Sensitive Editing Change History as a Programming Tool," *System Configuration Management: ECOOP '98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 146–157.

Auer, A., and J. Taramaa, "Experience Report on the Maturity of Configuration Management of Embedded Software," *Software Configuration Management: ICSE'96 SCM-6 Workshop*, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 187–197.

Ayer, S., and F. S. Patrinostro, *Documenting the Software Development Process: A Handbook of Structured Techniques*, New York: McGraw-Hill, 1992.

Ayer, S., and F. S. Patrinostro, *Software Configuration Management: Identification, Accounting, Control, and Management*, New York: McGraw-Hill,1992.

Baalbergen, E. H., K. Verstoep, and A. S. Tanenbaum, "On the Design of the Amoeba Configuration Manager," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 15–22.

Babich, W. A., *Software Configuration Management: Coordination For Team Productivity*, Boston, MA: Addison Wesley, 1986.

Bays, M. E., *Software Release Methodology*, NJ: Prentice Hall PTR, 1999.

Belanger, D., D. Korn, and H. Rao, "Infrastructure for Wide-area Software Development," *Software Configuration Management: ICSE'96 SCM-6 Workshop*, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (ed), Berlin: Springer-Verlag, 1996, pp. 154–165.

Bendix, L., "Fully Supported Recursive Workspaces," *Software Configuration Management: ICSE'96 SCM-6 Workshop*, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (ed), Berlin: Springer-Verlag, 1996, pp. 256–261.

Bendix, L., et al., "CoEd—A Tool for Versioning of Hierarchical Documents," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 174–187.

Ben-Menachem, M., *Software Configuration Guidebook*, London: McGraw-Hill International (UK) Limited, 1994.

Berczuk, S., and Appleton, B., *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, Boston, MA: Addison-Wesley, 2003.

Berlack, H. R., "Evaluation and Selection of Automated Configuration Management Tools," *Crosstalk: The Journal of Defense Software Engineering*, November 1995.

Berlack, H. R., *Software Configuration Management*, New York: John Wiley & Sons, Inc., 1992.

Berrada, K., F. Lopez, and R. Minot, "VMCM, A PCTE Based Version and Configuration Management System," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 43–52.

Bersoff, E. H., V. D. Henderson, and S. G. Siegel, *Software Configuration Management, An Investment in Product Integrity*, Englewood Cliffs, NJ: Prentice-Hall, 1980.

Bielikova, M., and P. Navrat, "Modeling Versioned Hypertext Documents," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 188–197.

Black, R., *Managing the Testing Process*, Redmond, Washington: Microsoft Press, 1999.

Blanchard, B. S., *System Engineering Management*, New York: John Wiley & Sons, 1991.

Bochenski, B., "Managing It All: Good Management Boosts C/S Success," *Software Magazine* Client/Server Computing Special Edition, Nov. 1993, p. 98.

Boehm, B. W., "A Spiral Model for Software Development and Enhancement," *IEEE Computer*, Vol.21, No.5, 1988, pp. 61–72.

Boehm, B. W., and P.N. Papaccio, "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering*, Vol. 14, No.10, 1988, pp. 1462–477.

Bohem, B. W., *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

Bouldin, B. M., *Agents of Change: Managing the Introduction of Automated Tools*, Englewood Cliffs: N.J., Yourdon Press, 1989.

Bounds, N. M., and S. Dart, "CM Plans: The Beginning to your CM Solution," Technical Report, Software Engineering Institute, Carnegie Mellon University, 1998.

Brereton, P., and P. Singleton, "Deductive Software Building," *Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops*, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 81–87.

Brooks F. P., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol.20, No.4, 1987, pp.10–19.

Brooks, F. P., *The Mythical Man-Month*, New York: Addison Wesley Longman, Inc., 1995.

Brown, A., et al., "The State of Automated Configuration Management," Technical Report, Software Engineering Institute, Carnegie Mellon University, 1991.

Brown, W. J., *Antipatterns and Patterns in Software Configuration Management*, New York: John Wiley & Sons, Inc., 1999.

Buckle, J. K., *Software Configuration Management*, Basingstoke: Macmillan, 1982.

Buckley, F. J., *Implementing Configuration Management: Hardware, Software, and Firmware*, Los Alamitos, Calif.: IEEE Computer Society Press, 1996.

Buffenbarger, J., "Syntactic Software Merging," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 153–172.

Buffenbarger, J., and K. Gruell, "What Have You Done for Me Lately? (Branches, Merges and Change Logs)," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 18–24.

Burrows, C., "Configuration Management: Coming of Age in the Year 2000," *Crosstalk: The Journal of Defense Software Engineering*, March 1999, pp. 12–16.

Burrows, C., and I. Wesley, *Ovum Evaluates: Configuration Management*, London: Ovum Limited, 1998.

Burrows, C., S. Dart, and G. W. George, *Ovum Evaluates: Software Configuration Management*, London: Ovum Limited, 1996.

Burton, T., "Software Configuration Management Helps Solve Year 2000 Change Integration Obstacles," *Crosstalk: The Journal of Defense Software Engineering*, January 1998, pp. 7–8.

Butler, T., et al., "Software Configuration Management: A Discipline with Added Value," *Crosstalk: The Journal of Defense Software Engineering*, July 2001, pp. 4–8.

Cagan, M., and D. W. Weber, "Task-Based Software Configuration Management: Support for 'Change Sets' in Continuus/CM," Technical Report, Continuus Software Corporation, 1996.

Cagan, M., "Untangling Configuration Management," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 35–52.

Caputo, K., *CMM Implementation Guide*, Reading, MA: Addison-Wesley Publishing Company, 1998.

Casavecchia, D. E., "Reality Configuration Management," *Crosstalk: The Journal of Defense Software Engineering*, November 2002, pp. 17–21.

Cave W. C., and G. W. Maymon, *Software Lifecycle Management: The Incremental Method*, Basingstoke: Macmillan, 1984.

Choi, S. C., and W. S. Scacchi, "Assuring the Correctness of Configured Software Descriptions," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 66–75.

Chris, A. "Why Can't I Buy an SCM Tool?" Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 278–281.

Christensen, A., and T. Egge, "Store—A System for Handling Third-Party Applications in a Heterogeneous Computer Environment," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 263–276.

Christensen, H. B., "Experiences with Architectural Software Configuration Management in Ragnarok," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 67–74.

Ci, J. X., et al., "ScmEngine: A Distributed Software Configuration Management Environment on X.500," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 108–127.

Clemm, G. M., "Replacing Version-Control with Job-Control," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 162–169.

Clemm, G. M., "The Odin System," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 241–262.

Coallier, F., "International Standardization in Software and Systems Engineering," *Crosstalk: The Journal of Defense Software Engineering*, February 2003, pp. 18–23.

Compton, S. B., and Conner, G. R., *Configuration Management for Software*, New York: Van Nostrand Reinhold, 1994.

Compton, S. B., and G. R. Conner, *Configuration Management for Software*, New York: Van Nostrand Reinhold, 1994.

Conradi, R., (ed.), Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings, Berlin: Springer-Verlag, 1997.

Conradi, R., and B. Westfechtel, "Configuring Versioned Software Products," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (ed), Berlin: Springer-Verlag, 1996, pp. 88–109.

Conradi, R., and B. Westfechtel, "Towards a Uniform Version Model for Software Configuration Management," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 1–17.

Conradi, R., and C. R. Malm, "Cooperating Transactions against the EPOS Database," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 98–101.

Continuus Software Corporation, "Change Management for Software Development," Continuus Software Corporation, 1998.

Continuus Software Corporation, "Distributed Code Management for Team Engineering," Continuus Software Corporation, 1998.

Continuus Software Corporation, "Problem Tracking and Task Management for Team Engineering," Continuus Software Corporation, 1998.

Continuus Software Corporation, "Software Configuration Management for Team Engineering," Continuus Software Corporation, 1998.

Continuus Software Corporation, "Task-Based Configuration Management: A New Generation Of Software Configuration Management," Continuus Software Corporation, 1997.

Cook, D. A., "Laws of Software Motion," *Crosstalk: The Journal of Defense Software Engineering*, February 2004, pp. 31.

Cook, D.A., "Software Process Improvement–A Good Idea for Other People," *Crosstalk: The Journal of Defense Software Engineering*, April 2004, p. 31.

Crnkovic, I., "Experience of using a Simple SCM Tool in a Complex Development Environment," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (ed), Berlin: Springer-Verlag, 1996, pp. 262–263.

Crnkovic, I., "Experience with Change-Oriented SCM Tool," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 222–234.

Crnkovic, I., and P. Willfor, "Change Measurements in an SCM Process," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 26–32.

Daniels, M. A., *Principles of Configuration Management*, Annandale, VA: Advanced Application Consultants Inc., 1985.

Dart, S., "Content Change Management: Problems for Web Systems," *Crosstalk: The Journal of Defense Software Engineering*, January 2000, pp. 1–7.

Dart, S., "Achieving the Best Possible Configuration Management Solution," *Crosstalk: The Journal of Defense Software Engineering*, September 1996.

Dart, S., "Adopting an Automated Configuration Management Solution," Technical Paper, STC'94 (Software Technology Center), Utah, 12 April, 1994.

Dart, S., "Best Practice for a CM Solution," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 239–255.

Dart, S., "Concepts in Configuration Management Systems," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 1–18.

Dart, S., "Concepts in Configuration Management Systems," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1994.

Dart, S., "Configuration Management Bibliography," Technical Report, Software Engineering Institute, Carnegie Mellon University, 1992.

Dart, S., "Containing the Web Crisis Using Configuration Management," Web Engineering Workshop at the Conference on Software Engineering (ICSE'99), May 16–17, 1999, Los Angeles, USA.

Dart, S., "Content Change Management: Problems for Web Systems" International Symposium on System Configuration Management SCM9, Toulouse France, September 5–7, 1999.

Dart, S., "Not All Tools are Created Equal," *Application Development Trends*, Vol.3, No. 9, 1996, pp. 45–48.

Dart, S., "Parallels in Computer-Aided Design Framework and Software De-velopment Environment Efforts," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1994.

Dart, S., "Past, Present and Future of CM Systems," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1992.

Dart, S., "Spectrum of Functionality in Configuration Management Systems," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1990.

Dart, S., "The Agony and Ecstasy of Configuration Management (Abstract)," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 204–205.

Dart, S., "Tool Configuration Assistant," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 110–113.

Dart, S., and J. Krasnov, "Experiences in Risk Mitigation with Configuration Management," *4th SEI Risk Conference*, November, 1995.

Dart, S., *Configuration Management: The Missing Link in Web Engineering*, Norwood, MA: Artech House, 2000.

Dart, S., To Change or Not to Change, *Application Development Trends*, Vol. 4, No. 6, 1997, pp. 55–57.

Dart, S., "WebCrisis.Com: Inability to Maintain," *Software Magazine*, September 1999.

Davis, A., and P. Sitaram, "A Concurrent Process Model for Software Development," *Software Engineering Notes*, Vol.19, No.2, pp. 38–51.

Davis, A. M., 201 *Principles of Software Development*, New York: McGraw-Hill, Inc., 1995.

Dehforooz, A., and F. J. Hudson, *Software Engineering Fundamentals*, New York: Oxford University Press, Inc., 1996.

DeMillo, R. A., et al., *Software Testing and Evaluation*, The Benjamin Cummings Publishing Company, Inc., 1987.

Deustsch, M. S., *Software Verification and Validation: Realistic Project Approaches*, Englewood Cliffs, NJ: Prentice Hall, 1982.

Dinsart, A., et al., "Object Derivation and Validation from a Data Base Definition," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 170–178.

Dix, A., T. Rodden, and I. Sommerville, "Modeling the Sharing of Versions," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 282–290.

DOD-STD-2167A–Defense System Software Development, 1988.

DOD-STD-2168–Defense System Software Quality Program, 1988.

Donald, R., D. R. Michels, and D. Bollinger, "Transitioning From SA-CMM to CMMI in the Special Operations Forces Systems Program Office," *Crosstalk: The Journal of Defense Software Engineering*, February 2002, pp. 12–14.

Donaldson, S. E., and S. G. Siegel, *Cultivating Successful Software Development: A Practitioner's View*, Upper Saddle River, NJ: Prentice Hall PTR, 1997.

Dyer, M., *The Cleanroom Approach to Quality Software Development*, New York: John Wiley & Sons, 1992.

Eggerman, W. V., *Configuration Management Handbook*, Blue Ridge Summit, PA: TAB Books, 1990.

Eidnes, H., D. O. Hallsteinsen, and D. H. Wanvik, "Separate Compilation in CHIPSY," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 42–45.

Eilfield, P., "Configuration Mangement as "Gluware" for Development of Client/Server Applications on Heterogeneous and Distributed Environments," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 264–271.

Estublier, J. (ed.), "System Configuration Management," *9th International Symposium, SCM-9 Toulouse, France, September 5–7, 1999 Proceedings*, Berlin: Springer-Verlag, 1999.

Estublier, J. (ed.), Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Berlin: Springer-Verlag, 1995.

Estublier, J., "Workspace Management in Software Engineering Environments," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 127–138.

Estublier, J., and R. Casallas, "Three Dimensional Versioning," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 118–135.

Estublier, J., J. Favre, and P. Morat, "Toward SCM/PDM Integration?" *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 75–94.

Estublier, J., S, Dami, and M. Amiour, "High Level Process Modeling for SCM Systems," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 81–97.

Evans, M. W., A. M., Abela, and T. Beltz, "Seven Characteristics of Dysfunctional Software Projects," *Crosstalk: The Journal of Defense Software Engineering*, April 2002, pp. 16–20.

Evans, M. W., *Productive Software Test Management*, Chichester, England: John Wiley & Sons, Inc., 1984.

Falkerngerg, B., "Configuration Management for a Large (SW) Development," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 34–37.

Feiler P. H., "Software Configuration Management: Advances in Software Development Environments" Technical Report, Software Engineering Institute, Carnegie Mellon University, 1990.

Feiler P. H., and G.F. Downey, "Tool Version Management Technology: A Case Study," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1990.

Feiler P. H., and G.F. Downey, "Transaction-Oriented Configuration Management: A Case Study," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1990.

Feiler, P. H., "Managing Development of Very Large Systems: Implications for Integrated Environment Architectures," Technical Paper, Software Engineering Institute, Carnegie-Mellon University, 1988.

Feiler, P. H., Configuration Management Models in Commercial Environments, Technical Report, Software Engineering Institute, Carnegie Mellon University, 1991.

Feiler, P. H., Software Configuration Management: Advances in Software Development Environments, Technical Paper, Software Engineering Institute, Carnegie-Mellon University, 1990.

Florence, A., "Reducing Risks Through Proper Specification of Software Requirements," *Crosstalk: The Journal of Defense Software Engineering*, April 2002, pp. 13–15.

Frohlich, P., and W. Nejdl, "WebRC: Configuration Management for a Cooperation Tool," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 175–185.

Gallagher, K., "Conditions to Assure Semantically Consistent Software Merges in Linear Time," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 80–83.

Gardy, R. B., *Successful Software Process Improvements*, NJ: Prentice Hall PTR, 1997.

Gentleman, W. M., S. A. MacKay, and D. A. Stewart, "Commercial Real-Time Software Needs Different Configuration Management," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 152–161.

Gilb, T., Principles of Software Engineering Management, New York: Addison-Wesley Publishing Company, 1988.

Gill, T., "Stop–Gap Configuration Management," *Crosstalk: The Journal of Defense Software Engineering*, February 1998, pp. 3–5.

Glass, R. L., *Software Runaways*, NJ: Prentice Hall, 1998.

Godart, C., et al., "About some Relationships between Configuration Management, Software Process and Cooperative Work: COO Environment," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 173–178.

Gowen, L. D., "Predicting Staff Sizes to Maintain Networks," *Crosstalk: The Journal of Defense Software Engineering*, November 2001, pp. 22–26.

Grosjean, S., "Building a CM Database: Nine Years at Boeing," *Crosstalk: The Journal of Defense Software Engineering*, January 2000, pp. 8–10.

Grossman, R., "Defect Management: A Study in Contradictions," *Crosstalk: The Journal of Defense Software Engineering*, September 2003, pp. 28–30.

Gulla, P., and J. Gorman, "Experiences with the use of a Configuration Language," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (ed), Berlin: Springer-Verlag, 1996, pp. 198–219.

Gustavsson, A., "Maintaining the Evolution of Software Objects in an Integrated Environment," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 114–117.

Hall, R. S., D. Heimbigner, and A. L. Wolf, "Requirements for Software Deployment Languages and Schema," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 198–203.

Haque, S., "Introducing Process into Configuration Management," *Crosstalk: The Journal of Defense Software Engineering*, June 1996.

Haque, T., "Process-Based Configuration Management: The Way to Go to Avoid Costly Product Recalls," *Crosstalk: The Journal of Defense Software Engineering*, April 1997.

Haque, T., "The F-16 Software Test Station Program: A Success Story in Process Configuration Management," *Crosstalk: The Journal of Defense Software Engineering*, November 1997.

Hass, A. M. J., *Configuration Management: Principles and Practice*, Boston, MA: Addison-Wesley, 2003.

Haug, M., et al., *Managing the Change: Software Configuration and Change Management*, Berlin: Springer-Verlag, 2001.

Hedin, G., L. Ohlsson, and J. McKenna, "Product Configuration using Object-Oriented Grammars." *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 107–126.

Heiman, R. V., and E. Quinn, Software Configuration Management Meets the Internet, Technical Report, Framingham, MA: International Data Corporation, November 1997.

Heiman, R. V., et al., Programmer Development Tools: 1997 Worldwide Markets and Trends, Technical Report, Framingham, MA: International Data Corporation, July 1997.

Heiman, R. V., S. Garone, and S. D. Hendrick, Development Life-Cycle Management: 1999 Worldwide Markets and Trends, Technical Report, Framingham, MA: International Data Corporation, June 1999.

Heiman, R. V., S. Garone, and S. D. Hendrick, Development Life-Cycle Management: 1998 Worldwide Markets and Trends, Technical Report, Framingham, MA: International Data Corporation, May 1998.

Heiman, R. V., The Growing Market for Software Configuration Management Tools, Technical Report, Framingham, MA: International Data Corporation, September 1997.

Heimbigner, D., and A. L. Wolf, "Post-Deployment Configuration Mangement," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 272–276.

Hoek, A., D. Heimbigner, D., and A. L. Wolf, "Does Configuration Management Research have a Future?" Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 305–309.

Hoek, A., D. Heimbigner, D., and A. L. Wolf, "System Modeling Resurrected," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer–Verlag, 1998, pp. 140–145.

Hoek, A., et al., "Software Deployment: Extending Configuration Mangement Support into the Field," *Crosstalk: The Journal of Defense Software Engineering*, February 1998, pp. 9–13.

Holdsworth, J., *Software Process Design: Out of the Tar Pit*, London: McGraw-Hill International (UK) Ltd., 1994.

Hunt, J. J., et al., "Distributed Configuration Management via Java and the World Wide Web," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 161–174.

Hunt, J. J., K. Vo, and W. F. Tichy, "An Empirical Study of Delta Algorithms," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 49–66.

IEEE Std-1028-

Humbly, J., and Farley, D., *Continuous Delivery*, Upper Saddle River, NJ: Addison-Wesley, 2011.

Humphrey, W. S., *Managing the Software Process*, New York: Addison-Wesley Publishing Company, 1989.

2008–IEEE Standard for Software Reviews and Audits, 2008.

IEEE Std-610.12-1990–IEEE Standard Glossary of Software Engineering Terminology, 1990.

IEEE Std-828-1998–IEEE Standard for Software Configuration Management Plans, 1998.

IEEE Std-828-2012–IEEE Standard for Configuration Management in Systems and Software Engineering, 2012.

IEEE, IEEE Software Engineering Standards Collection: 1999 Edition, New Jersey: IEEE, 1999.

Ince, D., *An Introduction to Software Quality Assurance and its Implementation*, London: McGraw-Hill International (UK) Ltd., 1994.

Ince, D., *ISO 9001 and Software Quality Assurance*, London: McGraw-Hill International (UK) Ltd., 1994.

Ingram, P., C. Burrows, and I. Wesley, *Configuration Management Tools: a Detailed Evaluation*, London: Ovum Limited, 1993.

Intersolv, Cost Justifying Software Configuration Management, PVCS Series for Configuration Management White Paper, Intersolv, Inc., 1998.

Intersolv, Software Configuration Management for Client/Server Development Environments: An architecture guide, White Paper, Intersolv, Inc., 1998.

Intersolv, Software Configuration Management: A primer for development teams and managers, White Paper, Intersolv, Inc., 1997.

ISO 10007: 1995–Quality Management—Guidelines for Configuration Management, 1995.

ISO 9000-3: 1997–Guidelines for the Application of ISO 9001:1994 to the Development, Supply, Installation And Maintenance of Computer Software, 1997.

ISO 9001: 1994–Quality systems—Model for Quality Assurance in Design, Development, Production, Installation and Servicing, 1994

Jackelen, G., "Verification and Validation People Can Be More Than Technical Advisors," *Crosstalk: The Journal of Defense Software Engineering*, February 2004, pp. 26–29.

Jacobson, I., M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process, and Organization for Business Success*, New York: ACM Press, 1997.

Jenner, M. G., *Software Quality Measurement and ISO 9001: How To Make Them Work for You*, New York: John Wiley & Sons, Inc., 1995.

Jones, C., *Software Quality: Analysis and Guidelines for Success*, London: International Thompson Press, 1997.

Jones, C. T., *Estimating Software Costs*, New York: McGraw-Hill. 1998.

Jones, G.W., *Software Engineering*, New York: John Wiley & Sons, 1990.

Jordan, M., "Experiences in Configuration Management for Modula-2," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 126–128.

Kaiser, G. W., "Modeling Configuration as Transactions," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 133–134.

Kasse, T., "Software Configuration Management for Project Leaders," Technical Paper, Institute for Software Process Improvement, Inc., Belgium, 1998.

Kasse, T., and P. A. McQuaid, "Factors Affecting Process Improvement Initiatives," *Crosstalk: The Journal of Defense Software Engineering*, August 2000, pp. 4–7.

Kelly, M. V., *Configuration Management: The Changing Image*, New York: McGraw-Hill, 1995.

Kenefick, S., *Real World Software Configuration Management*, Apress, 2003.

Keyes, J., *Software Configuration Management*, Boca Raton: Auerbach Publications, 2004.

Keyes, J., *Software Engineering Productivity Handbook*, New York: McGraw-Hill, 1993.

Kilpi, T., "Product Management Requirements for SCM Discipline," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 186–200.

Kinball, J., and A. Larson, "Epochs, Configuration Schema, and Version Cursors in the KBSA Framework CCM Model," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 33–42.

Kingsbury, J., "Adopting SCM Technology," *Crosstalk: The Journal of Defense Software Engineering*, March 1996.

Kirzner, R., Managing Content: The Key to Success in Web Business, Technical Report, Framingham, MA: International Data Corporation, June 1999.

Kolvik, S., "Introducing Configuration Management in an Organization," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 220–230.

Korel, B., et al., "Version Management in Distributed Network Environment," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 161–166.

Kramer, S. A., "History Management System," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 140–143.

Lacroix, M., and P. Lavency, "The Change Request Process," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 122–125.

Lacroix, S. M. J., D. Roelants, and J. E. Waroquier, "Flexible Support for Cooperation in Software Development," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 102–108.

Lago, P., and R. Conradi, "Transaction Planning to Support Coordination," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 145–151.

Lange, R., and R. W. Schwanke, "Software Architecture Analysis: A Case Study," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 19–28.

Larson, J., and H. M. Roald, "Introducing ClearCase as a Process Improvement Experiment," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer–Verlag, 1998, pp. 1–12.

Leblang, D. B., "Managing the Software Development Process with ClearGuide," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 66–80.

Leblang, D. B., and P. H. Levine, "Software Configuration Management: Why is it needed and What should it do?" Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 53–60.

Lee, T., P. Thomas, and V. Lowen, "An Odyssey towards best SCM Practices: The Big Picture," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 231–238.

Lehman, M.M., "Software Engineering, the Software Process and Their Support," Software Engineering Journal, Vol. 6, No. 5, 1991, pp. 243–258.

Lehman, M. M., and L. Belady, "A Model of Large Program Development," *IBM Systems Journal*, Vol. 15. No.3, 1976, pp. 225–252.

Lehman, M.M., and L. Belady, *Program Evolution: Processes of Software Change*, London: Academic Press, 1985.

Leishman, T. R., and D. A. Cook, "But I Only Changed One Line of Code!" *Crosstalk: The Journal of Defense Software Engineering*, January 2003, pp. 20–23.

Leishman, T. R., and Cook, D. A., "Lessons Learned From Software Engineering Consulting," *Crosstalk: The Journal of Defense Software Engineering*, February 2004, pp. 4–6.

Leishman, T. R., and Cook, D. A., "Requirements Risks Can Drown Software Projects," *Crosstalk: The Journal of Defense Software Engineering*, April 2002, pp. 4–8.

Leishman, T. R., and D. A. Cook, "Risk Factor: Confronting the Risks That Impact Software Project Success," *Crosstalk: The Journal of Defense Software Engineering*, May 2004, pp. 31–34.

Lie, A., et al., "Change Oriented Versioning in a Software Engineering Database," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 56–65.

Lientz, B. P., and E. B. Swanson, *Software Maintenance Management*, Reading, MA: Addison Wesley, 1980.

Lin, Y., and S. P. Reiss, "Configuration Management in terms of Modules," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 101–117.

Lindsay, P., and O. Traynor, "Supporting Fine-grained Traceability in Software Development Environments," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 133–139.

Lubkin, D., "Heterogeneous Configuration Management with DSEE," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 153–160.

Lundholm, P., "Design Management in Base/OPEN," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 38–41.

Lyn, F., *Change Control During Computer Systems Development*, Englewood Cliffs, NJ: Prentice Hall, 1991.

Lyon, D. D., *Practical CM*, Pittsfield, MA: Raven Publishing, 1994.

MacKay, S. A., "Changesets Revisited and CM of Complex Documents," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 277–281.

MacKay, S.A., "The State-of-the-art in Concurrent Distributed Configuration Management," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 180–193.

Mack–Crane, B., and A. Pal, "Conflict Management in a Source Version Management System," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 149–151.

Magnusson, B. (ed.), System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings, Berlin: Springer-Verlag, 1998.

Magnusson, B., and U. Asklund, "Fine Grained Version Control of Configurations in COOP/Orm," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 31–48.

Maraia, V., *The Build Master: Miscrosoft's Software Configuration Management Best Practices*, Upper Saddle River, NJ: Addison-Wesley, 2006.

Marciniak, J. J. (ed.), *Encyclopedia of Software Engineering (2nd Edition)*, New York: John Wiley & Sons, 2002.

Marshall, A. J., "Demystifying Software Configuration Management," *Crosstalk: The Journal of Defense Software Engineering*, May 1995.

Marshall A.J., "Software Configuration Management: Function or Discipline?" *Crosstalk: The Journal of Defense Software Engineering*, October 1995.

Martin, J., and C. McClure, *Software Maintenance: The Problem and its Solutions*, Englewood Cliffs, NJ: Prentice Hall, 1983.

Martin, J., *Rapid Application Development*, Englewood Cliffs, NJ: Prentice-Hall, 1991.

McCann, R. T., "How Much Code Inspection Is Enough?" *Crosstalk: The Journal of Defense Software Engineering*, July 2001, pp. 9–12.

McClure, C., *Software Reuse Techniques: Adding Reuse to the System Development Process*, NJ: Prentice Hall, 1997.

McClure, S., Web Development Life-Cycle Management Software, Technical Report, Framingham, MA: International Data Corporation, June 1999.

McConnel, S., *Code Complete (2nd Edition)*, Redmond, Washington: Microsoft Press, 2004.

McConnel, S., *Software Project Survival Guide*, Redmond, Washington: Microsoft Press, 1998.

McDermid, J. A., (ed.), *Software Engineer's Reference Book*, CRC Press, Inc., 1994

McDonald, J., P. N. Hilfinger, and L. Semenzato, "PRCS: The Project Revision Control System," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 33–45.

McKnight, W. L., "What Is Information Assurance?" *Crosstalk: The Journal of Defense Software Engineering*, July 2002, pp.4–6.

Meiser, K., "Software Configuration Management Terminology," *Crosstalk: The Journal of Defense Software Engineering*, January 1995.

Mell. P., and Grance, T., *The NIST Definition of Cloud Computing, National Institute of Standards and Technology*, US Department of Commerce, September, 2011.

Micallef, J., and G. M. Clemm, "The Asgard System: Activity-based Configuration Management," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp.175–186.

Mikkelsen, T., and S. Pherigo, *Practical Software Configuration Management: The Latenight Developer's Handbook*, Upper Saddle River, NJ: Prentice-Hall PTR, 1997.

Milewski, B., "Distributed Source Control System," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 98–107.

Miller, D. B., R. G. Stockton, and C. W. Krueger, "An Inverted Approach to Configuration Management," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 1–4.

Miller, T. C., "A Schema for Configuration Management," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 26–29.

MIL-STD-1521B–Technical Reviews and Audits for Systems, Equipment and Computer Programs, 1985.

MIL-STD-480B–Configuration Control Engineering Changes, Deviations and Waivers, 1988.

MIL-STD-481B–Configuration Control Engineering Changes (Short Form), Deviations and Waivers, 1988.

MIL-STD-482A–Configuration Status Accounting Data Elements & Related Features, 1974.

MIL-STD-483A–Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs, 1985.

MIL-STD-490A–Specification Practices, 1985.

MIL-STD-973–Configuration Management, 1995.

Molli, P., "COO-Transaction: Supporting Cooperative Work," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 128–141.

Moore, J. W., *Software Engineering Standards: A User's Road Map*, New Jersey: IEEE, 1997.

Mosley, V., et al., "Software Configuration Management Tools: Getting Bigger, Better, and Bolder," *Crosstalk: The Journal of Defense Software Engineering*, January 1996, pp. 6–10.

Munch, B. P., "HiCoV—Managing the Version Space," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 110–126.

Musa, J.D., "Software Engineering: The Future of a Profession," *IEEE Software*, Vol.22, No.1, 1985, pp. 55–62.

Narayanaswamy, K., "A Text-Based Representation for Program Variants," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 30–33.

NASA, NASA Software Configuration Management Guidebook, Technical Report SMAP-GB-A201, NASA, 1995.

Newbery, F. J., "Edge Concentration: A Method for Clustering Directed Graphs," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 76–85.

Nicklin, P. J., "Managing Multi-Variant Software Configuration," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 53–57.

Nierstrasz, "Component-Oriented Software Development," *Communications of the ACM*, Vol.35, No.9, 1992, pp.160–165.

Noll, J., and W. Scacchi, "Supporting Distributed Configuration Management in Virtual Enterprises," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 142–160.

Ochuodho, S. J., and A. W. Brown, "A Process-Oriented Version and Configuration Management Model for Communications Software," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 109–120.

Opperthauser, D., "Defect Management in an Agile Development Environment," *Crosstalk: The Journal of Defense Software Engineering*, September 2003, pp. 21–24.

Pakstas, A., "Aladdin/Lamp: Configuration Management Tools for Distributed Computer Control Systems," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989 ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 141–144.

Parker, K., "Customization of a Commercial CM System to Provide Better Management Mechanisms," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 289–292.

Peach, R. W. (ed.), *The ISO 9000 Handbook*, New York: The McGraw-Hill Companies, Inc., 1997.

Perry, D. E., "Dimensions of Consistency in Source Versions and System Compositions," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 29–32.

Perry, D. E., "System Compositions and Shared Dependencies," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 139–153.

Persson, A., "Experiences of Customization and Introduction of a CM Model," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 293–303.

Petersen, G., "The Weakest Geek," *Crosstalk: The Journal of Defense Software Engineering*, July 2001, pp. 31.

Pfleeger, S. L., *Software Engineering: Theory and Practice*, NJ: Prentice Hall, 1998.

Phillips, M., "CMMI Version1.1: What Has Changed?" *Crosstalk: The Journal of Defense Software Engineering*, February 2002, pp. 4–6.

Platinum, "Configuration Management and Software Testing," White Paper, Platinum Technology, Inc., 1999.

Platinum, "Controlling Application Development Costs using Software Configuration Management (CM)," White Paper, Platinum Technology, Inc., 1999.

Platinum, "How to Evaluate a CM Tool for Client/Server Environments?" White Paper, Platinum Technology, Inc., 1999.

Platinum, "Strategic Thinking About Software Change Management," White Paper, Platinum Technology, Inc., 1999.

Platinum, "The Expanding Role of Software Change and Configuration Management (CM)," White Paper, Platinum Technology, Inc., 1999.

Platinum, "What is CM?" White Paper, Platinum Technology, Inc., 1999.

Ploedereder, E., and A. Fergany, "The Data Model of the Configuration Management Assistant (CMA)," Second International Workshop: Software Configuration Management Proceedings, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 5–14.

Potter, N., and Sakry, M., "The Documentation Diet," *Crosstalk: The Journal of Defense Software Engineering*, October 2003, pp. 21–24.

Powers, J., *Configuration Management Procedures*, Santa Ana, CA: Global Engineering Documents, 1984.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (8th Edition)*, New York: McGraw-Hill, 2014.

Pressman, R. S., *Software Engineering: A Practitioner's Approach (5th Edition)*, New York: The McGraw-Hill Companies, Inc., 2001.

Rahikkala, T., J. Taramma, and A. Valimaki, "Industrial Experiences from SCM Current State Analysis," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 13–25.

Rawlings, J. H., *SCM for Network Development Environments*, New York: McGraw-Hill, 1994.

Ray, R. J., "Experiences with a Script-Based Software Configuration Management System," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 282–287.

Reichberger, C., "Orthogonal Version Management," Second International Workshop: *Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 137–140.

Reichenberger, C., "Delta Storage for Arbitrary Non–Text Files," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 144–152.

Reichenberger, C., "VOODOO—A Tool for Orthogonal Version Management," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 61–79.

Render, H., and R. Campbell, "An Object-Oriented Model of Software Configuration Management," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 127–139.

Reps, T., and T. Bricker, "Illustrating Interference in Interfering Versions of Programs," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 46–55.

Reuter, J., et al., "Distributed Version Control via the WWW," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 166–174.

Rice, R. W., "Surviving the Top 10 Challenges of Software Test Automation," *Crosstalk: The Journal of Defense Software Engineering*, May 2002, pp. 26–29.

Rich, A., and M. Solomon, "A Logic-Based Approach to System Modeling," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 84–93.

Rigg, W., C. Burrows, and P. Ingram, *Ovum Evaluates: Configuration Management Tools*, London: Ovum Limited, 1995

Rosenblum, D. S., and B. Krishnamurthy, "An Event-Based Model of Software Configuration Management," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 94–97.

Royce, W. "Managing the Development of Large Software Systems: Concepts and Techniques," IEEE, WESCON, 1970.

Royce, W., Software *Project Management: A Unified Framework*, Reading, MA: Addison Wesley Longman, Inc., 1998.

Rustin, R. (ed.), *Debugging Techniques in Large Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1971.

Samaras, T. T., and F. Czerwinski, *Fundamentals of Configuration Management*, Chichester, England: John Wiley & Sons, Inc., 1971.

Samaras, T. T., *Configuration Management Deskbook: Vol. 1*, Annandale, VA: Advanced Application Consultants Inc., 1988.

Samaras, T. T., *Configuration Management Deskbook: Vol. 2, Instruction Supplement,* Annandale, VA: Advanced Application Consultants Inc., 1988.

Schach, S. R., *Software Engineering*, Boston, MA: Richard D. Irwin, Inc., 1990.

Schmerl, B. R., and C. D. Maralin, "Designing Configuration Management Facilities for Dynamically Bound Systems," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 88–100.

Schmerl, B. R., and C. D. Maralin, "Versioning and Consistency for Dynamically Composed Configurations," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 49–65.

Schroeder, U., "Incremental Variant Control," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 144–148.

Schroeter, J., Marcel, M. P., and Lochau, M., "Dynamic Configuration Management of Cloud-based Applications," Proceedings of the 16th International Software Product Line Conference-Volume 2, September 2012, pp. 171–178.

Schulmeyer, G. G. and J. I. McManus, *Handbook of Software Quality Assurance*, London: International Thompson Press, 1996.

Schwanke, R. W., and M. A. Platoff, "Cross References are Features," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 86–95.

SEI, "A Quick Guide to Information about Software Environments, Configuration Management, and CASE," Technical Report, Software Engineering Institute, Carnegie Mellon University, 1995.

SEI, Carnegie Mellon University, The Capability Maturity Model: Guidelines for Improving the Software Process, Reading, MA: Addison Wesley Longman Ltd., 1994.

Seiwald, C., "Inter-file Branching—A Practical Method for Representing Variants," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 67–75.

Sheard, S. A., "How Do I Make My Organization Comply With Yet Another New Model?" *Crosstalk: The Journal of Defense Software Engineering*, February 2002, pp. 15–20.

Sheedy, C., "Sorceress: A Database Approach to Software Configuration Management," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 121–126.

Simmonds, I., "Configuration management in the PACT Software Engineering Environment," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 118–121.

Simmonds, I., "Duplicates': A Convention for Defining Configurations in PCTE-Based Environments," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 58–61.

Sodhi, J., and P. Sodhi, *Software Reuse: Domain Analysis and Design Process*, New York: McGraw-Hill, 1999.

Sommerville, I. (ed.), Software Configuration Management: ICSE'96 SCM 6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Berlin: Springer-Verlag, 1996.

Sommerville, I., *Software Engineering (9th Edition)*, Reading, MA: Addison-Wesley Publishing Company, 2011.

Sorensen, R., "The CM Database: To Buy or to Build?" *Crosstalk: The Journal of Defense Software Engineering*, January 2000, pp.11–12.

Sorensen, R., "CCB-An Acronym for "Chocolate Chip Brownies?" *Crosstalk: The Journal of Defense Software Engineering*, March 1999, pp. 3–6.

Starbuck, R. A., "A Beginner's Look at Process Improvement Documentation," *Crosstalk: The Journal of Defense Software Engineering*, March 2004, pp.18–20.

Starbuck, R., "A Configuration Manager's Perspective," *Crosstalk: The Journal of Defense Software Engineering*, July 2000, pp. 12–14.

Starbuck, R. A., "Software Configuration Management by MIL-STD-498," *Crosstalk: The Journal of Defense Software Engineering*, June 1996.

Starbuck, R. A., "Software Configuration Management: Don't Buy a Tool First," *Crosstalk: The Journal of Defense Software Engineering*, November 1997.

Starbuck, R. A., "Using CM to Recapture Baselines for Y2K Compliance Efforts," *Crosstalk: The Journal of Defense Software Engineering*, March 1999, pp. 7–11.

Thayer, R. H., and M. Dorfman (eds.), *Software Engineering Essentials (Volume I): The Development Process (Fourth Edition)*, Carmichael, California: Software Management Training Press, 2013.

Thayer, R. H., and M. Dorfman (eds.), *Software Engineering Essentials (Volume II): The Supporting Processes (Fourth Edition)*, Carmichael, California: Software Management Training Press, 2013.

Thayer, R. H., and M. Dorfman (eds.), *Software Engineering Essentials (Volume III): The Engineering Fundamentals (Fourth Edition)*, Carmichael, California: Software Management Training Press, 2013.

Thomas, I., "Version and Configuration Management On A Software Engineering Database," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 23–25.

Thompson, K., "People Projects: Psychometric Profiling," *Crosstalk: The Journal of Defense Software Engineering*, April 2003, pp. 18–23.

Thomson, R., and I. Sommerville, "Configuration Management Using SySL," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 106–109.

Tibrook, D., "An Architecture for a Construction System," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 76–87.

Tichy, W. F., *Configuration Managment*, New York: John Wiley & Sons, 1994.

Tryggeseth, E., B. Gulla, and R. Conardi, "Modeling Systems with Variability Using the PROTEUS Configuration Language," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 216–240.

Vacca, J., *Implementing a Successful Configuration Change Management Program*, [AU: city of publication?]Information Systems Management Group, 1993.

Van De Vanter, M. L., "Coordinated Editing of Versioned Packages in the JP Programming Environment," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 158–173.

Ventimiglia, B., "Effective Software Configuration Management," *Crosstalk: The Journal of Defense Software Engineering*, February 1998, pp. 6–8.

Viskari, J., "A Rationale for Automated Configuration Status Accounting," Software Configuration Management: ICSE SCM 4 and SCM 5 Workshops, (Selected Papers), Estublier, J. (ed.), Berlin: Springer-Verlag, 1995, pp. 138–144.

Vlokdijk, G., *Configuration Management 100 Success Secrets*, [AU:city of publication?]Emereo Publishing, 2008.

Wallnau, K. C., "Issues and Techniques of CASE Integration with Configuration Management," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1992.

Watts, F. B., *Engineering Documentation Control Handbook (2nd Edition)*, Park Ridge, NJ: Noyes Publications, 2000.

Watts, F. B., *Engineering Documentation Control Handbook: Configuration Management for Industry*, Park Ridge, NJ: Noyes Data Corporation/Noyes Publications, 1993.

Weatherall, B., "A Day in the Life of a PVCS Road Warrior: Want To Get PVCS Organized Quickly in a Mixed-Platform Environment?" Technical Paper, Synergex International Corporation, 1997.

Weber, D. W., "Change Sets Versus Change Packages: Comparing Implementations of Change-Based SCM," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp., pp.25–35.

Weber, D. W., "Change-Based SCM Is Where We're Going," Continuus Software Corporation, 1997.

Wein, M., et al., "Evolution is Essential for Software Tool Development," Position Paper, Institution of Information Technology, National Research Council Canada, 1995.

Weinberg, G. M., "Destroying Communication and Control in Software Development," *Crosstalk: The Journal of Defense Software Engineering*, April 2003, pp. 4–8.

Westfechtek, B., "Revision Control in an Integrated Software Development Environment," *Second International Workshop: Software Configuration Management Proceedings*, Princeton, NJ, October 1989, ACM Staff (ed.), New York: Association for Computing Machinery, 1989, pp. 96–105.

Westfechtel, B., "Structure-Oriented Merging of Revisions of Software Documents," *Proceedings of the 3rd International Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Staff (ed.), New York: Association for Computing Machinery, 1991, pp. 68–79.

Westfechtel, B., and R. Conradi, "Software Configuration Management and Engineering Data Management: Differences and Similarities," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 95–106.

Whitgift, D., *Methods and Tools for Software Configuration Management*, Chichester, England: John Wiley & Sons, Inc., 1991.

Wingerd, L., and C, Seiwald, "Constructing a Large Product with Jam," *Software Configuration Management: ICSE'97 SCM-7 Workshop Proceedings*, Boston, MA, May 1997, Conradi, R., (ed.), Berlin: Springer-Verlag, 1997, pp. 36–48.

Wingerd, L., and C. Seiwald, "High-level Best Practices in Software Configuration Management," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 57–66.

Yourdon, E., *Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*, NJ: Prentice Hall PTR, 1997.

Zahran S., *Software Process Improvement: Practical Guidelines for Business Success*, Harlow, England: Addison Wesley Longman Ltd., 1998.

Zave, P., "The Operational Versus Conventional Approach to Software Development," *Communications of the ACM*, Vol.27, No.2, 1992, pp.104–118.

Zeller, A., "Smooth Operations with Square Operations-The Version Set Model in ICE," Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 1996, (Selected Papers), Sommerville, I., (Ed), Berlin: Springer-Verlag, 1996, pp. 8–31.

Zeller, A., "Versioning System Models through Description Logic," *System Configuration Management: ECOOP'98 SCM-8 Symposium Proceedings*, Magnusson, B. (ed.), Berlin: Springer-Verlag, 1998, pp. 127–132.

Glossary and List of Acronyms

Abend: Abnormal end (i.e., termination of a process before completion).

Acceptance testing: Testing conducted to determine whether or not a system or product is acceptable to the customer, user, or client.

Adaptive maintenance: Software maintenance performed to make a computer program work in an environment different from the one that it was originally designed for.

AEA: American Electronics Association.

AIA: Aerospace Industries Association.

Allocated baseline: The initial approved specifications governing the development of configuration items that are part of a higher-level configuration item.

Allocated configuration identification: The current approved specifications governing the development of configuration items that are part of a higher-level configuration item.

Allocation: The process of distributing requirements, resources, or other entities among the components of a system or program.

ANSI: American National Standards Institute.

Application software: Software designed to fulfill specific needs of a user or client.

Approval: The agreement that an item is complete and suitable for its intended use.

Approved data: Data that has been approved by an appropriate authority and is the official (identified) version of the data until replaced by another approved version.

Archived data: Released or approved data that are to be retained for historical purposes.

Attributes: Performance, functional, and physical characteristics of a product.

Audit: An independent examination or review conducted to assess whether a product or process or set of products or processes are in compliance with specifications, standards, contractual agreements, or some other criteria.

Baseline: A specification or product that has been formerly reviewed and agreed upon and serves as a basis for further development; it can be changed only through change management procedures. Baselines can be defined at various parts of the development life cycle.

Baseline management: The application of technical and administrative direction to designate the documents and changes to those documents that formerly identify and establish baselines at specific times during the life cycle of a configuration item; in other words, the set of activities performed for establishing and maintaining the different baselines in a project.

Bug: An error, defect, fault or problem.

Build: The process (or the final result of the process) of generating an executable, testable, system from source code.

CA: Configuration audit.

CASE: Computer-aided software engineering. CASE is the use of computers to aid in the software engineering process. This can include the application of software tools to software design, requirements analysis, code generation, testing, documentation, and other software engineering activities.

CCA: Configuration control authority; another name for CCB.

CCB: Configuration control board (also known as change control board). A group of people responsible for evaluating and approving or disapproving changes to configuration items.

Certification: A written guarantee that a system or component complies with its specified requirements and is acceptable for operational use.

Change control: See configuration control

Change request form: A change request form is a form (paper or electronic) that is used to initiate a change and that contains the details of the change such as the name of the change originator, item to be changed, and details of changes.

CI: Configuration item. An aggregation of hardware or software or both that is designated for configuration management and treated as a single entity in the configuration management process.

CM: Configuration management.

CMM: Capability maturity model.

Configuration: The functional and/or physical characteristics of hardware or software items as set forth in the technical documentation and achieved in a product.

Configuration: (1) The performance, functional, and physical attributes of an existing or planned product, or a combination of products. (2) One of a series of sequentially created variations of a product.

Configuration audit: Product configuration verification accomplished by inspecting documents, products and records; and reviewing procedures, processes, and systems of operation to verify that the product has achieved its required attributes (performance requirements and functional constraints) and the product's design is accurately documented.

Configuration control: Configuration control is the element of configuration management consisting of the evaluation, coordination, approval or

disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification.

Configuration documentation: Technical information, the purpose of which is to identify and define a product's performance, functional, and physical attributes (e.g., specifications, drawings).

Configuration identification: The configuration management activity that encompasses selecting configuration documents; assigning and applying unique identifiers to a product, its components, and associated documents; and maintaining document revision relationships to product configurations.

Configuration management: A management process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design and operational information throughout its life.

Configuration status accounting: The configuration management activity concerning capture and storage of, and access to, configuration information needed to manage products and product information effectively.

Configuration verification: The action verifying that the product has achieved its required attributes (performance requirements and functional constraints) and the product's design is accurately documented.

Contract: As used herein, denotes the document (for example, contract, memorandum of agreement or understanding, purchase order) used to implement an agreement between a customer (buyer) and a seller (supplier).

Corrective maintenance: Maintenance that is performed to correct faults in software.

COTS software: Commercial off-the-shelf software

CPC: Computer program component.

CPCI: Computer program configuration item.

CR: Change request. A request to make a change or modification.

Crash: The sudden and complete failure of a computer system or component.

Criticality: The degree of impact that a requirement, module, error, fault or failure has on the development or operation of a system; synonymous with severity.

CSC: Computer software component. A functionally or logically distinct part of a computer software configuration item.

CSCI: Computer software configuration item. A software item that is identified for configuration management.

Data: Recorded information of any nature (including administrative, managerial, financial, and technical), regardless of medium or characteristics.

Datagram: See packet.

Debug: To detect, locate, and correct faults in a computer program.

Delta: A technique to store versions by storing only the differences between versions as opposed to storing each version in its entirety. Forward deltas store

the original version in its entirety and later versions as deltas. Reverse deltas store the most recent version in its full form and previous versions as deltas.

Design information: Technical information resulting from translating requirements for a product into a complete description of the product.

Design phase: The period of time in the software development life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements.

Design standard: A standard that describes the characteristics of a design or design description of data or program components.

Detailed design: The process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented.

Development testing: The testing conducted during the development of a system or component. This kind of testing is usually done independently in the development environment by the developer.

Developmental configuration: The software and associated technical documentation that define the evolving configuration of a computer software configuration item during development.

Disapproval: Conclusion by the appropriate authority that an item submitted for approval is either incomplete or not suitable for its intended use.

Document representation: A set of digital files that collectively represent a complete digital document

ECMI: European Computer Manufacturers Institute.

ECP: Engineering change proposal. A proposed engineering change and the documentation by which the change is described and suggested.

EIA: Electronics Industries Alliance.

Engineering change: An alteration in the configuration of a configuration item after formal establishment of its configuration identification.

EPRI: Electric Power Research Institute.

ERP: Enterprise resource planning.

ESA: European Space Agency.

Evaluation: The process of determining whether an item or activity meets specified criteria.

FAA: Federal Aviation Authority

Failure: The inability of a component or system to perform its required functions within specified performance requirements.

FCA: Functional configuration audit. An audit conducted to verify that the development of a configuration item has been completed satisfactorily, that the item has achieved the performance and functional characteristics specified in the functional and allocated configuration identification, and that its operational and support documents are complete and satisfactory.

Firmware: The combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device. The software cannot be readily modified under program control.

Fit: The ability of a product to interface or interconnect with or become an integral part of another product.

Form: The shape, size, dimensions, and other physically measurable parameters that uniquely characterize a product. For software, form denotes the language and media.

Form, fit, and function: The configuration comprised of the physical and functional characteristics of an item as an entity, but not including any characteristics of the elements making up the item.

Formal specification: A specification written and approved in accordance with established standards and guidelines.

Formal testing: Testing conducted in accordance with test plans and procedures that have been reviewed and approved by a customer, user, or designated approving authority.

FQR: Formal qualification review. The test or inspection by which a group of configuration items comprising a system is verified to have met specific contractual performance requirements.

FR: Fault report. Same as problem report.

Function: The action or actions that a product is designed to perform.

Functional attributes: Measurable performance parameters including reliability, maintainability, and safety.

Functional baseline: Functional baseline is the initial approved technical documentation for a configuration item. Also known as requirements baseline.

Functional specification: A document that specifies the functions that a system or component must perform.

Functional testing: The testing process that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. Also known as black-box testing.

Hardware: Devices or machines that are capable of accepting and storing computer data, executing a systematic sequence of operations on computer data, or producing control outputs. Such devices can perform substantial interpretation, computation, communication, control, or other logical functions.

HWCI: Hardware configuration item.

IDE: Integrated development environment. An environment where the user can design, develop, debug, test, and run the application is[AU:what is missing here?] developed. Some popular IDEs are Visual C++, PowerBuilder, and Delphi.

IEEE: Institute of Electrical and Electronics Engineers.

Incremental development: A software development methodology in which requirements definition, design, implementation, and testing occur in an

overlapping and interactive manner, resulting in incremental completion of the overall software product.

Informal testing: Testing conducted in accordance with test plans and procedures that have not been reviewed and approved by a customer, user or designated approving authority.

INPO: Institute of Nuclear Power Operations.

Inspection: A static analysis technique that relies on visual examination of development products to detect errors and faults.

Integration testing: The testing process where the software components, hardware components, or both are combined and tested to evaluate the interaction between them and how they perform in combination.

Integrity: The degree to which a system or component prevents unauthorized access to, or modifications of, computer programs or data.

Interchangeable: A product that possesses such functional and physical attributes as to be equivalent in performance to another product of similar or identical purposes and capable of being exchanged for the other product without selection for fit or performance, and without alteration of the products themselves or of adjoining products, except for adjustment.

Interface: A shared boundary across which information is passed.

Interface control: The process of identifying all functional and physical characteristics relevant to the interfacing of two or more configuration items provided by one or more organizations and ensuring that the proposed changes to these characteristics are evaluated and approved prior to implementation.

Interface documentation: Interface control drawing or other documentation that depicts physical, functional, and test interfaces of related or cofunctioning products.

Interface: The performance, functional, and physical attributes required to exist at a common boundary.

ISO: International Organization for Standardization.

IV&V: Independent verification and validation. Verification and validation that is performed by an organization that is technically, managerially, and financially independent of the development organization.

JCL: Job control language. A language used to identify a sequence of jobs, to describe their requirements to an operating system, and to control their execution.

Life cycle: A generic term relating to the entire period of conception, definition, build, distribution, operation, and disposal of a product.

Maintenance: The process of modifying a software system or component after delivery to correct faults, improve performance or another attribute, or adapt to a changed environment.

Metrics: Measures used to indicate progress or achievement.

MMI: Man-machine interface.

NASA: National Aeronautics and Space Administration.

NATO: North Atlantic Treaty Organization.

NIRMA: Nuclear Information & Records Management Association.

Nonconformance: [AU:pls check this entry]Nonfulfillment of specified requirement operational information, information that supports the use of a product (e.g., operation maintenance and user's manuals or instructions, procedures, and diagrams.

NOR: Notice of revision. A form used in configuration management to propose revisions to a drawing or list and, after approval, to notify users that the drawing or list has been, or will be, revised accordingly.

NSIA: National Security Industrial Association.

Object-oriented design: A software development technique in which a system or component is built using objects and connections between those objects.

Operational testing: The testing that is conducted to evaluate the performance of a system or component in its operational environment.

Original: The current design activity's document or digital document representation and associated source data file(s) of record (i.e., for legal purposes).

Patch: A modification made to a source program as a last-minute fix.

PCA: Physical configuration audit. An audit conducted to verify that a configuration item, as built, conforms to the technical documentation that defines it.

Perfective maintenance: Software maintenance performed to improve the performance, maintainability, or other attributes of a computer program.

Performance: A quantitative measure characterizing a physical or functional attribute relating to the execution of an operation or function. Performance attributes include quantity (how many or how much), quality (how well), coverage (how much area, how far), timeliness (how responsive, how frequent), and readiness (availability, mission or operational readiness). Performance is an attribute for all systems, people, products, and processes including those for development, production, verification, deployment, operations, support, training, and disposal. Thus, supportability parameters, manufacturing process variability, and reliability are all performance measures.

Performance specification: A document that specifies the performance characteristics that a system or component must possess.

Performance testing: Testing conducted to evaluate the compliance of a system or component with specified performance requirements.

Physical attributes: Quantitative and qualitative expressions of material features, such as composition, dimensions, finishes, form, fit, and their respective tolerances.

PR: Problem report. A report of a problem found in a software system or its documentation that needs to be corrected.

Preliminary design: The process of analyzing design alternatives and defining the architecture, components, interfaces, and timing and sizing estimates for a system or component.

Preventive maintenance: Maintenance performed for the purpose of preventing problems before they occur.

Product: Anything that is used or produced to satisfy a need (e.g., systems, hardware, software, firmware, data, processes, materials, or services).

Product baseline: The initial approved technical documentation (including source code, object code, and other deliverables) defining a configuration item during the production, operation, maintenance, and logistic support of its life cycle.

Product information: Information related to a product including configuration documentation and other information that is derived from configuration documentation (e.g., instruction manuals, manufacturing instructions, and catalogs).

Product support: The act of providing information, assistance, and training to install and make the software operational in its intended environment.

QA: Quality assurance. A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.

QC: Quality control. A set of activities designed to evaluate the quality of a developed product. In the case of QC, the focus is finding the defect, whereas in QA the focus is preventing the defect from occurring.

Query language: A language used to access information stored in a database.

Rapid prototyping: A type of prototyping in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process.

Regression testing: The process of testing a software system again usually using the original test plan and test data with the objective of ensuring that the modifications that were carried out have not caused unintended effects and that the system or component still complies with its specified requirements.

Release: A configuration management action whereby a particular version of software is made available for a specific purpose.

Released data: (1) Data that has been released after review and internal approvals, and (2) data that has been provided to others outside the originating group or team for use (as opposed to for comment).

Requirements analysis: The process of studying user needs to arrive at a definition of system, hardware, or software requirements.

Requirements phase: The period of time in the software development life cycle during which the requirements of a software product are defined and documented.

Requirements specification: A document that specifies the requirements for a system or component. Also known as the requirements definition document (RDD).

Requirements: Specified essential attributes.

Response time: The elapsed time between the end of an inquiry or command to an interactive computer system and the beginning of the system's response.

Retirement: Permanent removal of a system or component from its operational environment.

Retirement phase: The period of time in the software development life cycle during which the support for a software product is terminated.

Retrofit: The incorporation of new design parts or software code, resulting from an approved engineering change to a product's current approved product configuration documentation, into products already delivered to and accepted by customers.

Reusability: The degree to which a software component can be used in more than one computer program or system.

Rework: A procedure applied to a product to eliminate a nonconformance.

RFD: Requests for deviation.

RFW: Requests for waiver.

SAE: Society of Automotive Engineers.

SCM: Software configuration management. The set of methodologies, procedures, and techniques to manage and control change in the software development process and to ensure that products that are developed satisfy the requirements. IEEE defines SCM as a discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

SCM tool: Software that is used to automate SCM functions like change management, problem tracking, version management, build management, and status accounting that would otherwise have to be performed manually.

SCMP: Software configuration management plan. The SCM plan documents what SCM activities are to be done, how they are to be done, who is responsible for doing specific activities, when they are to happen, and what resources are required.

SCN: Specification change notice. A document used in configuration management to propose, transmit, and record changes to a specification.

SDLC: Software development life cycle. The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The SDLC typically includes different phases like analysis, design, development, testing, release, and maintenance.

Software: Computer programs, procedures, and associated documentation and data pertaining to the operation of a computer system.

Software development process: The software development process is that set of actions required for efficiently transforming the user's need into an effective software solution.

Software engineering: The application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. The practice of software engineering is a discipline, with a well-defined process (or system) that produces a product (e.g., the software and documentation) and has a set of (automated) tools for improving the productivity and quality of work.

Software product: The set of computer programs, procedures, and possibly associated documentation and data

Software program: A combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions.

Software tool: A computer program used in the development, testing, analysis, or maintenance of a program or its documentation.

Software unit: A separately compliable piece of code

Source code: Computer instructions and data definitions expressed in a form suitable for input to an assembler, compiler, or other translator.

Specification: A document that explicitly states essential technical attributes or requirements of a product and procedures to determine that the product's performance meets its requirements or attributes.

Spiral model: A model of the software development process in which the constituent activities, typically requirements analysis, preliminary and detailed design, coding, integration, and testing are performed interactively until the software is complete.

SPR: Software problem report. Same as problem report.

SQA: Software quality assurance. The discipline of applying quality assurance principles to the software development process.

SQAP: Software quality assurance plan. The document where the procedures and guidelines for practicing software quality assurance are recorded.

Submitted data: Released data that have been made available to customers.

Support equipment: Equipment and computer software required to maintain, test, or operate a product or facility in its intended environment.

System testing: Testing conducted on the complete system to evaluate whether it meets the specified requirements.

Unit: One of a quantity of items (e.g., products or parts)

Unit testing: The testing of individual units in a software system.

U.S. DOD: United States Department of Defense.

Usability: The ease with which a user can learn to operate, prepare inputs, and interpret outputs of a system or component.

Variance; deviation; waiver; departure: A specific written authorization to depart from a particular requirement(s) of a product's current approved configuration documentation for a specific number of units or a specified time period. (A variance differs from an engineering change in that an approved engineering change requires corresponding revision of the product's current approved configuration documentation, whereas a variance does not.)

Variant: Versions that are functionally equivalent but designed for different hardware and software environments.

Verification: The act of validating that a requirement has been fulfilled.

Version: Version is an initial release or rerelease of a configuration item. It is an instance of the system that differs in some way from the other instances. New versions of the system may have additional functionality or different functionality. Their performance characteristics may be different or they may be result of fixing a bug that was found by the user or customer.

Version identifier: A supplementary identifier used to distinguish a changed body or set of computer-based data (software) from the previous configuration with the same primary identifier. Version identifiers are usually associated with data (such as files, databases, and software) used by, or maintained in, computers.

Waterfall model: A model of the software development model in which the different phases, requirements, analysis, design, coding, testing, and implementation are performed in that order, without any overlap or interaction.

Working data: Data that have not been reviewed or released; any data that are currently controlled solely by the originator including new versions of data that were released, submitted, or approved.

WWW: World Wide Web.

About the Author

Alexis Leon is the cofounder and managing director of L & L Consultancy Services Pvt. Ltd., a company specializing in software engineering, Web design and development, groupware and workflow automation, software procedures, management and Industrial engineering, Internet/intranet development, and client/server application development.

He graduated from Kerala University with first rank and distinction in industrial engineering in 1989. In 1991, he earned his master's degree (M.Tech.) in industrial engineering with distinction also from the same university. His areas of specialization include software engineering, database management systems, workflow automation, groupware, ergonomics, and industrial engineering.

He has written more than 50 books on various computing, IT, and management topics. Two of these books have been translated into Mandarin by McGraw-Hill, Taiwan. Many of his books are prescribed textbooks in various universities and training institutes in India, Sri Lank, Nepal, Taiwan, China, Nigeria, and Ethiopia.

He is a senior member of Institute of Electrical and Electronics Engineers (IEEE) and member of the Association of Computing Machinery (ACM).

Before starting his own company, he has worked with Pond's India Ltd. as an industrial engineer, with Tata Consultancy Services as a software consultant, and with Cybernet Software systems as the technical director. He can be reached through his Web site at www.alexisleon.com.

Index