Example Software Development Process.

The example software development process is shown in Figure A. The boxes represent the software development process kernels. The Software Unit Testing, Software Component testing, Software Configuration Item Testing, Validation Test and Verification and Validation Test Design are the kernels that will be studied in detail in this course. The following paragraphs and frames will discuss each kernel and the test-related activities that go on during each kernel. The discussions on these kernels are to be considered general guidelines and are determined on the project size basis. The kernel concept and process was demonstrated by Humphrey in Chapter 13 of the referenced book. The Entry, Task, Verification, and Exit (ETVX) paradigm is a concept initially developed by a group at IBM, Radice et al.

The Example Software Development Process shown in Figure A is based on a predefined repository of process "kernels" from which the testing, verification & validation life cycle for a given project can be defined. A "kernel" is defined for each function such as Requirements Analysis, Document Review, Code Analysis, Unit Testing, etc. Each "kernel" contains entry criteria, inputs, activities, exit criteria, outputs, process controls, and metrics are defined for each kernel.

Entry Criteria describe the circumstances under which a kernel becomes activated. All entry criteria should be fulfilled before commencing with the activities defined for the kernel. If some entry criteria cannot be fulfilled, a work-around may be necessary. All such deviations from what is prescribed in the kernel must be performed to maximize risk reduction and minimize adverse impacts to quality. All deviations must also be documented appropriately.

Inputs identify the data items that are required to support the activities of the kernel. For the most part, these are outputs of other kernels or products of the software development process such as test plans or design documents.

Activities describe a minimum set of actions that will produce the output items and meet the exit criteria objectives. For each related set of actions, step by step procedures are available to support consistency among analysts, adherence to proven practices, and training. If all activities cannot be performed, management steps to reduce risk should be taken, they should be noted in the outputs products (such as the Requirements Analysis Report), and the kernel closed.

Exit Criteria identify the circumstances under which the kernel is completed or de-activated. It includes delivery or presentation of results, and passing of information to other kernels (such as the passing of comments to the Configuration Management kernel for tracking).

Outputs identify products of the kernel activities and are either deliverable items or are required to support other kernels.

Process Controls define quality assurance activities that are performed for the kernel. These are detailed in the Project Management and Quality Assurance kernels and are documented in the IV&V Project Management Plan.

Metrics are the categories of measures collected and maintained for each kernel. The details of each metric are specific to each kernel and are defined in a Metrics Program Plan. The metrics allow the monitoring of trends and identification of problem areas.

This kernel concept was used by Lillian K. Zelinski at Science Applications International Corporation (SAIC) in Arlington, VA and presented in her paper "Constructing Independent Verification and Validation Life Cycles Using Process Kernels", at the 10th Annual IEEE COMPASS Conference, June 26-30 1995, Gaithersburg, MD USA.

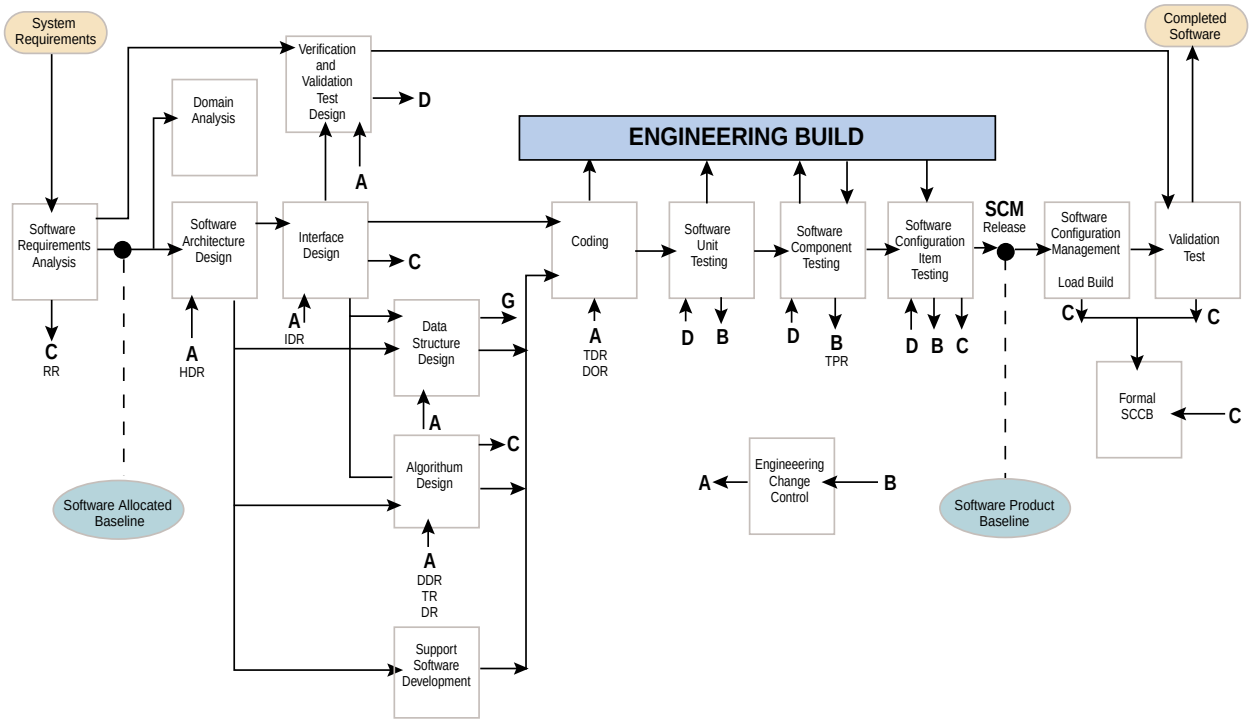

Figure A. Example Software Life Cycle Kernel Model

Figure B is a graphic of five different software life cycle models. The four other software life cycle models are Waterfall; Rapid Prototype; Incremental; Evolutionary and Spiral. The kernels in Figure A were presented in a straight waterfall model, however each process in Figure B has basically the same set of kernels. These kernels can be mapped to the development model (Waterfall, Cyclic, etc.) determined by the needs of the project.

Step 1 - Software Requirements Analysis Phase Kernel.

This kernel involves the further derivation of the system requirements that have been allocated to software. This kernel is included for all software development projects. Requirements provide a clearly stated, verifiable, and testable foundation for Software Engineering. Guidelines must be provided by management and followed when specifying and identifying requirements. The requirements clearly define the capabilities and

performance of the end software product.

For data-driven or data-intensive systems, the requirement specification activities also address data sources, types, and rates. The requirements are managed throughout the development process. The requirement analysis represents an agreement at the beginning of the software development process between the developers and the customer on what the delivered software product will be at completion.

This is just the beginning of this activity. Requirement review & analysis are conducted throughout the life cycle as requirements change or as new requirements are added. The primary responsibility of the testers in this phase is to ensure that requirements are testable.

Step 2 - Domain Analysis Kernel

An analysis is performed to identify existing non-developmental software (NDS) components. NDS may take the form of existing, reusable software or COTS software. A make versus reuse (including modifications) cost decision is made by the software engineering team. Software reuse also may occur at the design level. As part of the analysis, existing designs that may be adapted or modified should be identified. No activities are conducted here which are directly related to testing.

Step 3 - Verification & Validation Test Design Kernel

This kernel includes the definition of unit, software component, and software configuration item test cases, and data used by Software and Test Engineering to verify that the product is working as expected. These tests include functional tests, out-of-bounds tests, static and dynamic stress tests, and limit tests. Validation testing is acceptance testing of the software by or for the customer. The task involved in this kernel shall include the creation of software test plans, methods, descriptions, and procedures. The amount, type, and formality of testing are determined by requirements for security, size of development effort, and complexity of algorithms and data structures. The Software & Test Engineering will establish traceability between the requirement analysis products and the validation tests. This kernel starts refining some of the test planning activities started during the system planning. The software is better understood by this time and the Software & Test Engineering needs to fine tune the: test tools, drivers, stubs, simulators, stimulators, emulators and the types of data that are needed. The term Verification test will be used for phases of test planned and conducted by Software Engineering. Validation test will be used for phases of test planned and conducted by Test Engineering.

The phases listed here are the ones that will be discussed in the rest of the course units.

Software & Test Engineering

- Test phase planning: tools & data
- Software Engineering: verification test planning
- Unit test
- Software component test (integration)
- Software configuration item test

- Regression test

Test Engineering: validation test planning
- Function & system tests
- Installation & acceptance tests
- Regression test
- Outputs: phase-level test plans

Step 4 - Software Architecture Design Kernel.

This kernel is the high-level design of the software. It includes the definition of the software components and their structure and interaction. The software components, which are to be developed, are identified in this kernel. The System, Software, and Test Engineering have the responsibilities to analyze requirements in response to change and produce testable requirements and a disclaimer list if needed.

Step 5 - Interface Design Kernel

This kernel involves the early definition of the interfaces between each of the software units. It also includes the definition of interface external to the software (e.g., hardware and user dependencies) and software parameters such as data type and structure definitions. The System, Software, and Test Engineering team are responsible to identify the software units in this kernel and phase of the development phase. The System, Software, and Test Engineering have the responsibilities to analyze requirements in response to change and produce testable requirements and a disclaimer list if needed.

Step 6 - Data Structure Design Kernel

This kernel represents the detailed data structure design. The internal file structures, relationships, and data formats are defined either graphically or through a design language. The data design should conform to third normal form optimized for performance. If the chosen data design language uses the same syntax as the implementation language, the Software Manager must ensure that premature coding is not used to describe the design. No directly related test activities are conducted in this kernel. Another contractor or the customer at the kernel may conduct an independent verification and validation function of the data structure design.

Step 7 - Algorithm Design Kernel

This kernel represents the detailed design of the software logic and is included for all software development projects that implement control structures and/or algorithms. It includes the generation of the program design language (PDL) or other representation of the design, such as graphical representation methods. If the chosen PDL uses the same syntax as the implementation language, the Software Manager must ensure that premature coding is not used to describe the design. No activities are conducted here which are directly related to test.

Step 8 - Support Software Development Kernel

This kernel represents the detailed design and coding of all support software, including prototyping, modeling, and test-tool development. Test-tool development consists of all models, simulation, stimulation, and/or emulation software required to fully test and qualify the deliverable software. Depending on the required formality for test-tool development, this kernel may use any or all of other defined kernels.

Step 9 - Coding Kernel

This kernel is the creation of source code for the software units that implement the software design. Coding is done uniformly across the software products using a defined standard or guideline. A software guidelines and standards manual should be used for products implemented in Ada, C/C++, Java etc. The responsibilities of Software Engineering are to establish and design unit test cases, develop unit test drivers and stubs. The responsibilities of Test Engineering are to design test, develop test cases and identify the test data to use in the cases.

Step 10 - Software Unit Testing Kernel

This kernel involves execution of the unit test cases defined as part of the verification / validation test design. Unit testing is conducted for all developed software units. The number of tests required is driven by the complexity of the code. Methods such as McCabe's complexity metric should be used to uniformly determine the complexity and corresponding number of paths through the software. This testing may be accomplished in the host or target environment. Higher level testing, such as software component or software configuration item testing, is not used to fulfill unit testing. An entire set of units is dedicated to unit testing. Unit Testing is not normally a verification and validation activity, but is an important testing activity.

The responsibilities of Software Engineering is to:

- Execute test cases & log results
- Resolve defects
- Design & generate integration test plan(s) & test cases
- Outputs: test logs/reports, known defect log, & integration test documentation

The responsibilities of Test Engineering is to:

- Develop test cases
- Develop test tools
- Outputs: test cases

Step 11 - Software Component (Integration) Testing Kernel

The Software Component (Integration) Testing Kernel involves the execution of the software component test cases defined as part of the verification test design. The goal of this test is to verify the performance of the component and its internal (unit to unit) interfaces. This testing may be accomplished in the host or target environment. This kernel may be excluded for projects where the software size or complexity does not warrant additional verification

testing, and where the software configuration item test kernel is used in lieu of this kernel. An entire set of units is dedicated to software integration testing, so only briefly review is presented here.

The responsibilities of Software Engineering is to:
- Execute test cases & log results
- Resolve defects
- Document test logs and reports
- Document known defect in a report.

The responsibilities of Test Engineering is to:
- Finalize test cases
- Document test cases

Step 12 - Software Configuration Item Testing Kernel

The Software Configuration Item Testing Kernel involves the execution of the software configuration item prior to full integration testing with other software and hardware configuration items. The Software Engineer performs testing to verify that the software configuration item works as intended in the target environment.

For projects requiring stand-alone validation of the software, this test may be the dry run of the validation procedure. Hardware/software integration may occur at any level required supporting testing and is specified in the Test Plan. This time period should also be used by Test Engineering to conduct dry run activities, which are discussed in detail in later lessons.

The responsibilities of Test Engineering is to:
- Execute dry run of function & system tests Log defects
- Document corrected test cases
- Document defect report

The responsibilities of System & Software Engineering is to:
- Assist in resolving defects from dry run
- Conduct integration test of configuration items

Step 13 - Software Configuration Management Load Build

The Engineering Load Build Kernel involves the creation of the executable load builds from the configuration management engineering library to support software component integration and software configuration item validation testing. The environment used to create these loads and the procedures to be followed is under configuration control. These procedures and scripts are used to start the system. Software Configuration Management Load Build is refereed to as Cold Start procedures. This kernel may be excluded for software development projects where executable loads are built from software configuration management controlled libraries.

The responsibilities of Software Engineering is to:

- Integration of the executable load builds
- Implementation of the executable load builds
- Script for the executable load builds

The responsibilities of the Software Configuration Management
- Identification of the executable load builds from the engineering Library
- Control of the executable load builds from the engineering Library

Step 14 - Validation Test Kernel.

For software intensive systems or for projects where the validation of software as a stand-alone configuration item is required, the software validation kernel is used. In this case, the cold start of the software source code will be done as part of the software development activity. These phases of testing are dedicated to later lessons, so only a brief review is presented here.

The responsibility of Test Engineering is to:
- Execute function & regression test
- Execute system & regression test
- Execute installation & regression test
- Execute acceptance test
- Log defects
- Document corrected test cases
- Document defect reports
- Document test reports

The responsibility of System & Software Engineering is to Assist in resolving defects

Step 15 - Engineering Change Control Kernel

Change control can occur anywhere within the system development activity. During software development, the level of change control authority depends on the level of maturity of the product. During early development stages, the user, working through the user library, has control of the software products. As products mature to the verification test and higher integration levels, control transitions to the engineering library and engineering management control. The Software Configuration Management (SCM) release point is where software products become baselined (the point at which products transition to the SCM library and formal change control). This point can be changed to meet the needs of individual projects. Products are baselined at some point prior to validation testing. This kernel is depicted in a single phase, but in reality the activities associated with this kernel are conducted throughout the life cycle of the project.

The responsibility of Configuration Management is to
- Baseline test documents
- Provide controlled builds to Test Engineering
- Outputs: software CM load builds

The responsibility of System, Software, & Test Engineering is to:

- Requirements analysis in response to change
- Outputs: testable requirements, disclaimer list, & updated documentation & code

Step 16 - Formal Software Configuration Control Board.

The Formal Software Configuration Control Board Kernel involves change control. Change control can occur anywhere within the software development process. During software development, the level change control authority depends on the level of maturity of the product. During early development stages, the user, working through the user library, has control of he software products. As products mature to the verification test and higher integration levels, control transitions to the engineering library and engineering management control. The software configuration management release point is where software products become baselined. The baseline is the at which products transition to the software configuration management library and formal change control. The baseline can be changed to meet the needs of individual projects. The products must be baselined at some point prior to validation testing. The software configuration management process is another course in itself.

This completes the lecture on the Example Software Development Process and the contents of the kernels. The student should have an understanding of the kernel concept, the criteria of each kernel and how they can be applied to various software life cycle development processes.