Requirements Management with Use Cases

Module 2: Introduction to RMUC
RMUC: Course Outline

0 - About This Course
1 - Best Practices of Software Engineering
2 - Introduction to RMUC
3 - Analyze the Problem
4 - Understand Stakeholder Needs
5 - Define the System
6 - Manage the Scope of the System
7 - Refine the System Definition
8 - Manage Changing Requirements
9 - Requirements Across the Product Lifecycle
Introduction to RMUC: Overview

- Needs
- Features
- Software Requirements
- Test Procedures
- Design
- User Docs
- Problem Space
- Solution Space

The Product To Be Built
Introduction to RMUC: Module Objectives

- Define requirements management terms
- Gather information about why requirements management is crucial to project success
- Learn how requirements management drives the entire development process
- Begin use-case modeling of requirements
Why Are We Here?

The **GOAL** is to deliver

**quality** products

**on time and on budget**

which meet the customer’s

**real needs.**
Agreement on What the System Should Do

The Goal

Customer
User Community

Requirements
Verification

System
To Be Built

Surrogate
Goal

Requirements

Adapted from Al Davis
definitions: requirements and their management

- a requirement is a condition or capability to which the system must conform

- requirements management is a systematic approach to
  - eliciting, organizing, and documenting the requirements of the system, and
  - establishing and maintaining agreement between the customer/user and the project team on the changing requirements of the system
Requirements Specifications

Needs
↓
Features
↓
Software Requirements
↓

Vision Document

Use-Case Model

Supplementary Specifications

Test Specifications

Design Specifications

User Documentation Specifications
What Do We Do in Requirements Management?

- Analyze the Problem
  - [New System]
  - [Existing System]
  - [Incorrect problem]
  - [Addressing correct problem]

- Understand Stakeholder Needs
  - [New Input]
  - [Can't do all the work]
  - [Work in scope]

- Define the System
  - [More Iterations]
  - [Requirements Definition Complete]

- Manage the Scope of the System
  - [Requirements Definition Complete]

- Refine the System Definition

Requirements Management with Use Cases v2000
Copyright © 1998, 2000 Rational Software, all rights reserved
Workers and Artifacts in The Requirements Workflow

Diagram:

- Requirements Management Plan
- Glossary
- Staleh older Requests
- Use Case
- Use-Case Specifier
- Software Requirements Specification
- Use-Case Package
- System Analyst
- Vision
- Use-Case Model
- Actor (human)
- Boundary Class
- User-Interface Designer
- User-Interface Prototype
- Use-Case Storyboard
- Supplementary Specification
- Requirements Attributes
Linking Requirements

- Design & Implementation
- Project Management
- Change Management
- QA & Test

Attributes

Traceability
What Factors Contribute to Project Success?

**Project Success Factors**

<table>
<thead>
<tr>
<th>Factor</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) User Involvement</td>
<td>15.9%</td>
</tr>
<tr>
<td>2) Executive Management Support</td>
<td>13.9%</td>
</tr>
<tr>
<td>3) Clear Statement of Requirements</td>
<td>11.8%</td>
</tr>
</tbody>
</table>

However...

- Only 16% of projects completed on time and on budget (all companies)
  - 9% (large companies)
  - 28% (small companies)
- 53% of projects over-ran their original estimates
  - Average overrun: 189% (+$59 billion)
- 31% of projects canceled before completion ($81 billion)

**Why are these results so bad?**

Standish Group, '95 (www.standishgroup.com)
### What Factors Contribute to Project Failure?

#### Project Challenged (over-runs) Factors
1. Incomplete Requirements (13.1%)
2. Lack of User Involvement (12.4%)
3. Lack of Resources (10.6%)

#### Project Impaired (canceled) Factors
1. Incomplete Requirements (13.1%)
2. Lack of User Involvement (12.4%)
3. Lack of Resources (10.6%)

---

Standish Group, '95 (www.standishgroup.com)

---

<table>
<thead>
<tr>
<th>Factor</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Lack of User Input</td>
<td>12.8%</td>
</tr>
<tr>
<td>2) Incomplete Requirements and Specifications</td>
<td>12.3%</td>
</tr>
<tr>
<td>3) Changing Requirements and Specifications</td>
<td>11.8%</td>
</tr>
</tbody>
</table>

---

Requirements Management with Use Cases v2000
Copyright © 1998, 2000 Rational Software, all rights reserved
How Can Use Cases Help Projects Succeed?

- Show why the system is needed
  - Use cases: what users will be using the system for
  - Actors: who/what wants to use the system
- Provide unifying thread for system development
  - Use cases are the behavior the system must provide
  - Use cases drive Analysis & Design, and Test

The idea behind use cases is to decide what the system will be used for before defining what the system is supposed to do.
What Is a Use-Case Model? A Sample System

An Automated Teller Machine (ATM)

- Bank Customer
- Cashier
- Bank Consortium
- Maintenance Crew

Activities:
- Withdraw Cash
- Transfer Funds
- Deposit Funds
- Maintain ATM
**Definitions and symbols: Actors and Use Cases**

**Actor:** someone/something outside the system that interacts with the system

**Use case:** sequence of actions performed by a system that yields an observable result of value to an actor
Definitions and symbols: Communicates-Association

**Communicates-association (Line or Arrow):** an association between an actor and a use case, indicating that they interact.

**Arrow:** indicates who initiates communication with the use case.
Each Communicates-Association Is A Whole Dialog

Insert Card
Approve card
Enter PIN
Approve PIN
Enter account, amount

Dispense cash
Ask for Receipt
Printed receipt
Return bank card

Transmit request
Send back approval

Bank Customer
Withdraw Cash
Bank Consortium
A Use-Case Model Is Mostly Text!

Use-Case-Model Survey
- survey description
- list of all actors
- list of all use cases

Withdraw Cash
- brief description
- flow of events

Transfer Funds
- brief description
- flow of events

Deposit Funds
- brief description
- flow of events

Maintain ATM
- brief description
- flow of events

An ATM

Bank Customer

Bank Consortium

Cashier

Maintenance Crew

Bank Customer

Withdraw Cash

Transfer Funds

Deposit Funds

Maintain ATM
Sample System: Requested Features for An ATM

- Design the software for an automated teller machine (ATM). The ATM requests necessary information from the bank by communicating with the bank consortium system.
- The ATM accepts a cash card, interacts with the user via a display and a key pad, communicates with the bank consortium system to carry out the transaction, receives and dispenses cash, and prints out a receipt.
- A user should be able to withdraw cash, transfer funds between two of her accounts, and deposit to an account.
- When a transaction is finished, the card is ejected.
- The system requires appropriate record keeping and security provisions.
Sample System: Sketch of User Interface
1. Look at ATM Use-Case Model in the Handouts
   - Use-Case-Model Survey for ATM in Handout UC 1
   - Use-Case Report for Withdraw Cash in Handout UC 3.1
   - Don’t try to understand all the details. We’ll revisit later.

2. Answer the following questions
   - What actors are humans? What actors are not human?
   - How many use cases are there?
   - If a Bank Customer withdraws cash, is a receipt always printed?
   - If a Bank Customer wants to withdraw cash and enters a wrong PIN, what is this ATM required to do?
Benefits of Use Cases

❖ **Use cases are easy to understand**
  - Use terminology that customers and users understand
  - Verify developer understanding

❖ **Use cases give context for requirements**
  - Identify users of the system
  - Identify system interfaces
  - Put system requirements in logical sequences
  - Help verify that all requirements are captured

❖ **Use cases facilitate agreement with the customer on system requirements**
A Use-Case Model Helps with ‘Scope Creep’

- The customer is aware there is a change to the use-case model
- Makes the cost impact visible to the customer

I’ll need a new use case and change some existing use cases...
The High Cost of Requirement Errors

The 1-10-100 Rule

Relative Cost to Repair Errors: When introduced vs. when repaired

Boehm 1988

Average cost ratio 14:1
Grady 1989

“All together, the results show as much as a 200:1 cost ratio between finding errors in the requirements and maintenance stages of the software lifecycle.”

Boehm 1988
Involve the Whole Team in Requirements

- Developers, Testers, Writers
  - Help develop requirements management practices
  - Monitor adherence to practices
  - Verify elicitation process
  - Help document requirements
  - Participate in requirements reviews
  - Participate in or chair a CCB (Change Control Board)
  - Review traceability outcomes
  - Verify quality, testability, completeness
Review: Introduction to RMUC (Modules 0 and 2)

1. What is our goal?

2. How can we “measure” quality?
   - What are the FURPS categories of requirements?
   - What categories within each of these apply to your product?

3. Why is it important to establish a baseline? How is it done?

4. How can a Use-Case Model help a project succeed?
   - What is a use case? What is an actor?
   - What is the relationship from an actor to a use case?
   - How does a Use-Case Model help avoid scope creep?

5. What does the “1-10-100 Rule” tell us about finding defects?

6. What are some ways to involve the development team early?