Software Quality Engineering for Managers—Training Materials
Abstract: This document contains presentation and instructional materials for a four-hour seminar on the essential elements for developing high-quality, reusable software. The seminar includes sessions on process organization, managing software development, the software requirements specification (SRS) document, software reviews and inspections, software configuration management (SCM), and testing. The target audience for this course includes senior and middle managers, and project and quality managers with little or no software quality engineering background. These training materials were developed at the Semiconductor Equipment Technology Center (SETEC) at Sandia National Laboratories.

Keywords: Software Process Improvement, Software Development, Software Reliability, Training

Approvals: Harvey Wohlwend, Project Manager
Dan McGowan, Technical Information Transfer Team Leader
## Table of Contents

1 EXECUTIVE SUMMARY ........................................................................................................ 1
  1.1 Introduction ....................................................................................................................... 1
  1.2 Process Organization ......................................................................................................... 1
  1.3 Project/Risk Management ................................................................................................. 1
  1.4 Requirements ..................................................................................................................... 1
  1.5 Reviews and Inspections (Formal In-Process Reviews) .................................................... 1
  1.6 Configuration Management ............................................................................................. 1
  1.7 Testing ............................................................................................................................... 2
  1.8 Support .............................................................................................................................. 2

2 COURSE SYLLABUS .............................................................................................................. 2
  Course Schedule: ....................................................................................................................... 2
  Course Instructors: ..................................................................................................................... 2
  Target Audience: ........................................................................................................................ 2
  Objectives: ................................................................................................................................. 2
  Course Material: ......................................................................................................................... 2

3 SCHEDULE ................................................................................................................................ 2

4 OUTLINE .................................................................................................................................. 3
  Introduction ................................................................................................................................ 3
  Process Organization ................................................................................................................. 3
  Project/Risk Management ........................................................................................................ 3
  Requirements ............................................................................................................................. 4
  Reviews and Inspections (Formal In-Process Reviews) ............................................................ 4
  Configuration Management ..................................................................................................... 4
  Testing ........................................................................................................................................ 4
  Support ....................................................................................................................................... 4

5 INTRODUCTION ...................................................................................................................... 5

6 SOME WORKING DEFINITIONS ............................................................................................. 17

7 PROCESS ORGANIZATION ................................................................................................... 18

8 PROJECT / RISK MANAGEMENT .......................................................................................... 30

9 REQUIREMENTS ..................................................................................................................... 40

10 REVIEWS AND INSpections ............................................................................................... 47

11 CONFIGURATION MANAGEMENT ....................................................................................... 65

12 FORMAL SOFTWARE CONFIGURATION MANAGEMENT AND TIME-PHASED
   VERSION (BASELINE) RELEASE ............................................................................................... 72
  Overview .................................................................................................................................. 72
  Structure ................................................................................................................................... 72
  Change Request/Problem Reporting ....................................................................................... 74
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Control/Configuration Management</td>
<td>75</td>
</tr>
<tr>
<td>Software Engineering Activities</td>
<td>77</td>
</tr>
<tr>
<td>Summary</td>
<td>77</td>
</tr>
<tr>
<td>GLOSSARY OF SOFTWARE CONFIGURATION MANAGEMENT TERMINOLOGY</td>
<td>78</td>
</tr>
<tr>
<td>TESTING</td>
<td>81</td>
</tr>
<tr>
<td>SUPPORT</td>
<td>90</td>
</tr>
<tr>
<td>QUESTIONS TO ASK</td>
<td>98</td>
</tr>
<tr>
<td>ACTION PLAN FOR SOFTWARE QUALITY</td>
<td>99</td>
</tr>
<tr>
<td>SEI MATURITY MODEL: LEVEL 2 – REPEATABLE PROCESS</td>
<td>101</td>
</tr>
<tr>
<td>FOLLOW-UP QUESTIONS</td>
<td>104</td>
</tr>
<tr>
<td>SOME FURTHER READING</td>
<td>105</td>
</tr>
<tr>
<td>Articles Specific to Sessions</td>
<td>105</td>
</tr>
<tr>
<td>On Software Project Management</td>
<td>105</td>
</tr>
<tr>
<td>On Software Metrics</td>
<td>105</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1  SCM Context ..................................................................................................................73
Figure 2  Normal Release Cycling ..............................................................................................74

List of Tables

Table 1  Change Profiles .............................................................................................................75
Table 2  Contents Of Change Request Analysis .........................................................................76
Table 3  Contents of Change Implementation Package ...............................................................76
1 EXECUTIVE SUMMARY

This document contains presentation and instructional materials for a seminar on the essential elements necessary to develop high-quality, reliable software. Included are several key questions that managers and executives can (and should) ask about a software development project. A small case study is used to illustrate important elements in the software development process and as a focus for discussion. The seminar is structured as follows:

1.1 Introduction

The seminar begins with an introduction and a query of participants’ expectations. Although course goals are presented, emphasis is placed on areas of common interest to the class. Software is defined and two views of software quality are offered. The following sessions all relate to activities that denote a level of software engineering maturity that is well defined. Target audience includes project and quality managers with little or no background in software quality engineering.

1.2 Process Organization

This session focuses on the importance of viewing software development and software engineering from a process perspective. Software life cycle activities and their interrelationships are discussed, with emphasis on devoting sufficient time to pre-implementation activities. Also discussed are opportunities for quality improvement and use of some core metrics to track current processes and measure improvement.

1.3 Project/Risk Management

This session addresses a reasonable approach to the management aspects of software development. It teaches that each participant in a software development project must be aware of these activities, accompanying management planning, and what management planning does for the project.

1.4 Requirements

This session describes and discusses a software requirements specification (SRS) as the most important product definition document in software development. The reasons for an SRS and the content of an SRS also are presented.

1.5 Reviews and Inspections (Formal In-Process Reviews)

This segment covers the nature and purpose of software reviews, which satisfy both management and technical aspects of software development. Levels and types of reviews are discussed, with appropriate life cycle review points highlighted.

1.6 Configuration Management

Software configuration management (SCM) is an essential aspect of the software development process. This session covers the elements and benefits of SCM, and answers common questions about SCM.
1.7 Testing
Regardless of the care taken in developing software, the need remains to test software at all levels to validate it to requirements and ensure that it is as defect-free as possible before delivery to the customer. This session stresses the concept of testing to detect defects. Test approaches and activities for testing are presented and discussed.

1.8 Support
This concluding session recaps the elements of software engineering that promote high quality and high reliability. The session and the course is concluded by reviewing the software engineering Capability Maturity Model (CMM).

2 COURSE SYLLABUS

Course Schedule:
8:30 a.m. to 12:30 p.m.

Course Instructors:
Eric Tomlin, Sandia National Laboratories, Primary
Dave Peercy, Sandia National Laboratories, Alternate

Target Audience:
Senior and middle managers. Project and quality managers with little or no software quality engineering background.

Objectives:
• Develop a framework for software engineering (and the software life cycle).
• Recognize the benefits of integrating software engineering techniques into a design for quality and reliability.
• Outline an action plan for the manager to apply toward increasing the quality of software produced.

Course Material:
1. Course Notebook
2. Case Study Material - (Handouts at Seminar)

3 SCHEDULE
Seminar Length: 4 hours
8:30 AM
Introduction
4 OUTLINE

Introduction

The course is introduced along with a query of the class participant's expectations. Although there are course goals presented, emphasis will be placed on areas of common interest to the class. A working definition of software is presented as well as two views of software quality. The software engineering CMM from the Software Engineering Institute (SEI) is introduced to cause the participants to consider where their organizations might be in their ability to consistently produce high quality software. The following sessions all relate to activities that denote a level of software engineering maturity that is well defined.

Process Organization

This session focuses on the importance of viewing software development and software engineering from the perspective of a process. Software life cycle activities and their interrelationships are discussed with an emphasis on sufficient time spent on activities prior to implementation. Opportunities for quality improvement are discussed along with some core metrics that can be used to track current processes and measure improvement. This session starts a series of questions that managers could (and should) ask about software projects under their direction. An outline to action plan items also begins with this session.

Project/Risk Management

This session addresses a reasonable approach to the management aspects of software development. Each participant in a software development project must be aware of these activities, the accompanying management planning, and what management planning does for the project.
**Requirements**

An SRS is the most important product definition document in software development. The SRS is described and discussed in this session. The reasons for an SRS and the content of an SRS are also presented.

**Reviews and Inspections (Formal In-Process Reviews)**

Reviews satisfy both management and technical aspects of software development. This session covers the nature and purpose of software reviews. Levels of reviews and types of reviews are discussed with appropriate life cycle review points highlighted.

This session focuses on one particularly important type of review, the software inspection. It covers the background of software inspections, discusses pros and cons of inspections, and presents reported information of its return on investment. This session also describes the inspection process and roles and responsibilities of participants.

**Configuration Management**

SCM is an essential aspect of the software development process. This session covers the elements of SCM, benefits of SCM, and answers some common questions about SCM.

**Testing**

Regardless of the care taken in developing software, there is still a need to test software at all levels to validate the software to requirements and to ensure that the software is as defect-free as it can be before delivery to the customer. This session stresses the concept of testing to detect defects. Test approaches and activities for testing are presented and discussed.

**Support**

This concluding session recaps the elements of software engineering that promote high quality, high reliability software. The session and the course is concluded by revisiting the software engineering Capability Maturity Model.
Who Are We? What do We Do?

- Sandia National Laboratories
  - Operated by Martin Marietta for the Department of Energy - "... in the national interest."
  - Part of the Nuclear Weapons Complex
    - High Reliability - High Confidence Products
    - Extensive use of embedded software
  - Research extends to robotics, materials science, energy sources, computational mechanics, cryptography and many other areas

- Semiconductor Equipment Technology Center (SETEC)
  - Identification of cross-section of Laboratory assets to provide assistance to SEMATECH and the SEMI/SEMATECH member companies
What Do Managers Do?

"Management's job is to create an environment"
Why are YOU here?
Why is There a Problem?

Software as % of Development Work

Defects: When Created - When Found

Adapted from Software Logistics National Workshop, August 15-16, 1989
Seminar Objectives

• Develop a framework for software engineering
  – Basis for: Tracking cost & schedule, (Software) System Development Plan

• Recognize the benefits of integrating software engineering concepts, tools and techniques into a design for quality and reliability
  – Reduces risk, improves visibility into the schedule, and improves the quality of delivered products

• Outline an action plan for YOU to apply to increase the probability of producing high quality software
  – Results can be part of a Software Quality Plan and Software Development Plan
Seminar Format

- Problem/Dilemma
- Discussion
- Presentation on Prepared Material
- Questions Encouraged!
What is software?

"Ah, it's only code!"

- **Code**
  - Source, Objects, Executables

- **Documentation (Product Definition)**
  - Comprehensive written description - Defines requirements content, composition, design, performance, testing, use and support. Includes plans as appropriate (e.g., development, test, configuration management).
Software Quality

Depends on One's Perspective!

• If I have to use it . . .

• If I'm the one who has to fix it . . .
Software Quality

User Perspective

Usability

The effort required to learn, operate, prepare input and interpret output of a program.

Safety

The software design contains positive measures to prevent or reduce to an acceptable level of risk, system failures that could cause injury or death, or serious damage to critical equipment.

Efficiency

The extent to which software performs its intended functions with a minimum of computing resources (e.g., Response Time).

Reliability

The probability that software will not cause the failure of a system for a specified time under specified conditions.
Software Quality
Maintainer's Perspective

Maintainability

The ease with which software can be changed
- "Supportability"

Portability

The ease with which software can be transferred from one computer system or environment to another
Implies Reusability
Where is YOUR Organization?
(In its Goal to Produce High Quality Software)

Software Engineering Capability

Software Engineering Institute (SEI), Carnegie Mellon University

Maturity Level

5 - Optimizing
   [Feedback]

4 - Managed
   [Quantitative]

3 - Defined
   [Qualitative]

2 - Repeatable
   [Intuitive]

1 - Initial
   [Ad Hoc]

Problem Areas Addressed

Acquired Capabilities
- Adaptable to Change
- Automation
- Process Control
- Statistical Process Control
- Changing Technology
- Problem Prevention
- Process Measurement
- Fault Projection
- Test & Review Coverage
- Process Metrics Database
- Process Definition
- Quality Point Reviews
- Software Engineering Training

Results

Risk

Productivity & Quality

* Regression Testing: Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements
### 6 SOME WORKING DEFINITIONS

**Baseline**
A baseline is the documented identification of a software product—its code and all its related documentation at some specific point in time. It is the basis for all Software Configuration Management activities.

**Error**
A human action that results in one or more faults being inserted into a work product.

**Failure**
A departure of a computer program's operation from the user's requirements.

**Failure Intensity**
The number of failures occurring in a given time period.

**Fault**
Defect in the software that can cause a failure. Defect and fault are terms that are often interchanged.

**Inspection**
a formal evaluation of work products or software elements (requirements, design, code, etc.) to detect faults, violations of development standards and other problems. Participants have defined roles, minutes recorded and retained, results reported.

**Review**
Any number of techniques where a work product, or a set of work products, are presented for comment or approval. Reviews include management reviews.

**Regression Testing**
Selective retesting of a system or component to verify that modifications have not caused unintended effects and the system of component still complies with its specified requirements.

**Risk**
The probability (chance) of an unwanted event or outcome.

**Version**
A version is a software product with a defined set of capabilities. A new version is a variation of the previous version, in that it has a change in its functionality or performance characteristics. An example of this is if the system is changed to include an additional process or new feature.
Session Goals

- To establish a framework for the remaining sessions

- To discuss the importance of viewing software engineering from the perspective of a process

- To discuss quality improvement in light of the software engineering process
Relationship of Process Activities

**Requirements**
- Preliminary Design
- Integration Test Planning
- System Test Planning
- System Testing
- Integration Testing
- Unit Testing
- Unit Test Planning
- Detailed Design

**Production / Support (Users / Customers)**

**Users / Customers**
Embedded Software Life Cycle Model

Determine Objectives, Alternatives, Constraints

Evaluate Alternatives, Identify & Resolve Risks

- Risk Analysis - Cost, Schedule, Technical
- Design Tradeoffs
- Policy/Proc. for SW Quality
- Gen. SW Dev. Approach
- Explore Concept/ Customer Input
- Rqmts Analysis, HW-SW Tradeoffs
- Beta System

Plan Next Phases

- New Features:
  - Fold into Prototype, Beta.
  - or Production System depending on size and complexity
- Refine Software Development Plan
- Software System Test Planning
- Hardware/Software Integration Test Planning
- Software Integration Test Planning
- Refine Software Requirements
- Requirements Inspection
- Design Review/Inspection
- Code Inspection
- System Test
- Integration and Test
- Acceptance Test
- Test Plan Inspection/Review

Develop, Verify Next-Level Products

Process (Model) Definition - Why Bother?

- All work is a process
  - Has input(s) & output(s)

- Definition is needed to baseline process
  - Provides framework for defining development activities
  - Provides the foundation for measuring the process

- Definition required for repeatability
  - Points to areas of process for improvement
Software Quality Metrics
Measuring the Process and Its Products

Management (Resources)
Project vs History
Planned vs Actual (Cost—CONC*, Schedule Milestones, Program Size)

Process
% of Product-Definition Documentation Inspected
% of Product-Definition Documentation Under Configuration Management
% of Requirements Implemented à Tested (Test Coverage)
Defect Detection/Correction Rates

Product
- Correctness Faults per thousand lines of code - Fault (Defect) Density
- Reliability Failure Intensity (Failures/Hr of Operation) + MTBF**
- Availability Operational Uptime
- Supportability $$$; Percentage of Original Cost to Produce

* CONC - Cost of Non Conformance
** MTBF - Mean Time Between Failures
Software Quality Metrics

- Three views—Collecting the right measure at the right level . . . and for the right purpose

  - Strategic  à  Customer Based
  - Tactical  à  Project Management
  - Process & Product  à  Engineering

- Integrating lower level measures into the appropriate view
Process Definition

Foundation for Quality Improvement

Relies On:

- Focus on process, rather than the product
- Repeatability of process
- Measurement

Measuring degree of improvement against some standard

Emphasis on continuous improvement
Process Organization Summary

- Software development can (and must) be approached in a disciplined, engineering manner

- Key to improving software quality is to improve the process used to develop software

- A structured approach alone doesn't guarantee quality improvement
  - We need a little more than just the framework! We need to fill in the activities

- Question you should ask
  - Have we really defined OUR process?
Process Organization Summary

Action Plan

Make an appointment with your software manager to sketch out the structure of YOUR software development process:

- Use blank Embedded Life Cycle Model as basis or develop simpler model
- Compare to model in this course
- Ask: Is there elements of the course model that we want to implement?
  If so: When can we begin to implement them? What can we do in the next 6 months? In the next year?
Embedded Software Life Cycle Model

- Determine Objectives, Alternatives, Constraints
- Evaluate Alternatives, Identify & Resolve Risks
- Plan Next Phases
- Develop, Verify Next-Level Products

Session Goals

- To understand that planning, and documenting the planning, reduces risks in software development

- To discuss appropriate levels of software planning

- To show types of planning documents for software development
What Do We Need To Plan For?

What Activities Influence Quality?

- Requirements Analysis, Design, Code, and Testing
- Reviews and Software Inspections (Formal In-Process Reviews)
- Configuration Management and Change Control
- Prototyping and Beta Testing
- Support Procedures
- Standards, Practices and Conventions Regarding Development and Support
Planning Documents

• Software Quality Plan
  - List covered software products (scope of plan)
  - Define minimum documentation (product definition) requirements
  - Define minimum review requirements
  - Identify or reference review/inspection criteria
  - Specify standards, practices, and conventions
  - Specify minimum testing levels - required test types
  - Define configuration management policies
  - Require product-level plans as necessary

Institute of Electrical and Electronics Engineers (IEEE) Stds 730-1989 and 983-1986

Only 25% of SEMI/SEMATECH Companies have Quality Plans for Software

Planning Documents
(continued)

Software Development Plan (Project Management Plan)

-- Project management - process model, organizational structure
-- Management objectives - Risk management
-- Technical processes
-- Work packages, schedule, budget

Scheduling experience
Software Development Planning

• Project Management
  - Define Process Model - Major milestones
  - How are the groups organized within this framework? - Distinct hardware/software groups? - Some/all highly integrated?

• Management Objectives - Risk Management
  - Requirements, Schedule, Budget priorities
  - Risks - Tradeoffs: Contractual and Technological

• Technical Processes
  - Tools - Techniques: Analysis and Design Methods - Computer-Aided Software Engineering (CASE) tools

• Work Packages, Schedule, Budget
  - Dependency of work packages - PERT\(^*\), Gantt Charts
  - Cost and Schedule Models - COCOMO\(^*\) based good starting point

\(^*\) PERT Program Evaluation and Review Technique
\(^*\) COCOMO COnstructive COst MOdel
Planning Documents  
(Concluded)  

- Software Configuration Management Plan – Managing the software products  
  - Identify software products; control, implement, report change  
  - Change authority responsibilities (Configuration Control Board)  
  - Support tools and techniques  

- Test Plan (Software Portion)  
  Software Integration, Hardware - Software integration, System
What Affects Plan Content?

• Your own corporate quality assurance policy

• Internal engineering procedures & guidelines

• IEEE* standards and guides

• SETEC* - SEMATECH guides

• Common sense
  – Can use less than standard
  May need more than standard

• Information can be combined in one document

* IEEE Institute of Electrical and Electronics Engineers
* SETEC Semiconductor Equipment Technology Center (Sandia National Laboratories)
Risk Management Summary

• Planning is necessary to effectively and consistently produce quality software - Reduces Risk

• Adapt a standard approach to your particular environment - Document this approach

• Software managers and developers need your interest and support to plan effectively

• Questions you should ask
  – What is our policy for software quality?
  – Do we have standards for software development? - What are they?
  – Do we have a Software Development Plan? - Is it current?
  – Do we have a cost and schedule estimate?
  – How are we tracking to the development schedule?
Risk Management Summary
Action Plan

• Write out a policy statement defining YOUR company's concept of quality for software – Such as:
  (Sample Policy Statement)
  - "Quality software shall be an objective of this company's quality program.
   This objective applies to the acquisition, development, use, and support of software" "The program shall address, as a minimum:
   Organization, tasks, and responsibilities; software documentation; configuration management;
   testing, and; reviews and audits."

• Make an appointment with your software manager to outline software development goals for your next new feature (or next generation of equipment)
  - Integrate Process Model from previous action item - Technical risks - Analysis and Design methods - Initiate formal schedule and budget estimation, define work packages
Session Goals

• To understand what a Software Requirements Specification is

• To understand the importance of requirements analysis and the Software Requirements Specification in software development (and support)
What is a Software Requirements Specification?

WHAT:

- Contains a complete description of what is to be done, not how to do it
  
  A Software Requirements Specification is **not** a Design Document

- An agreement between Supplier and Customer (Top level - Partnering for Total Quality issue), and, System and Software Developers (Low level - Requirements allocation to embedded software)

- The most important document describing the software

... and WHY:

- Protects both software developer (or supplier) and system developer (or customer) from misunderstanding the other's intention

- How can "correctness" be determined without a specification?

- Serves as the baseline for subsequent development and verification activities
Why Bother with Requirements?

25-80% of faults traceable to errors in defining the requirements

Only 23% of SEMI/SEMATECH companies always write requirements

Cost of Fix a Fault
(Given the fault was created during requirements time frame)

- During Reqmts: $ 195
- During Design: 489
- During Coding: 997
- During Test: 7,136
- During Support: Even more!

Roger Fujii, Logicon, Inc.
Software Requirements Specification

- What does an SRS contain?
  - General Description of the Required Software
  - Constraints, Assumptions, and Dependencies
  - Specific Requirements
  - Supporting Information

  - An industry-wide consensus document
  - Recommended outline for an SRS
  - Templates available (Design and System Test templates also!)
Requirements Summary

- **SRS is the MOST important software document**
  - Embodies the results of requirements analysis

- **SRS specifies WHAT is to be done, but NOT HOW**

- **Questions you should be asking**
  - Do we know what the requirements are?
  - What analysis techniques are we using to define/refine requirements?
  - Are the requirements documented?
  - Who is our customer? - Who is the user?
  - Have we drafted the System Test Plan?

- **Don't ask!**
  - How much code have you written?
Requirements Summary
Action Plan

- Meet with (interview) customer production managers.
  Ask: What are their top five requirements for your company's equipment.

- Meet with your system/project and software managers
  Ask: How are we meeting these requirements?
Reviews and Inspections
Session Goals

- Identify characteristics of a review
- Discuss typical life cycle review points/types of reviews
- Convince you that formal reviews are important
Software Reviews

• What is reviewed?
  - Software elements or project status
    Product - Plans - Processes
    Schedule - Cost - Resources Available

• Why are reviews held?
  - Identify discrepancies (As they occur! - Instead of finding them later,
    e.g., during system test or letting the customer find them)
    Defects - Non-conformance - Inconsistencies/Ambiguities

• What are the results of the reviews?
  - Recommendations for improvements
    Prioritize - Ownership - Validate with follow-up review (as necessary)
<table>
<thead>
<tr>
<th>Reviews &amp; Inspections</th>
<th>4 12/94</th>
</tr>
</thead>
</table>

Typical Software Life Cycle Review Points

- Requirements Review
- Design Review
- Code Review
- Test Review
- Release Review
Requirements Reviews

- Technical project review of Software Requirements Specification – Baseline
- Formal Review (Inspection) of Software Requirements Specification for discrepancies
- Management walkthrough with customer

SUCCESS TIP * Project success depends on a good understanding of the requirements

Only 18% of SEMI/SEMATECH Companies always formally review Requirements - 8% never do
Design Review

- Technical project review of Software Design Document
  - Critical Design Review; Baseline
- Peer walkthrough of module design
- Inspection of Software Design Document for defects
- Management walkthrough with customer

Only 19% of SEMI/SEMATECH Companies always hold Design Reviews - 3% never do
Code Review

- Peer walkthrough of modules
- Peer walkthrough of unit test plan/results
- Inspection of source code for defects
  - Strongly Recommend!

Only 8% of SEMI/SEMATECH Companies always formally Inspect Code - 29% never do
Test Reviews

Peer walkthrough or Inspection of Test Plans and Results

Only 5% of SEMI/SEMATECH Companies always formally review Test Planning - 40% never do

Status Reviews

- Transition points - Minimum
- Planned vs Actual
  Cost, schedule, and technical progress
- Results of technical reviews
  Technical processes in control - Too many/Too few discrepancies
Software Inspections*
(Formal In-Process Reviews)

• Definition:
  – A formal evaluation technique in which software products
    (requirements, design, code, etc.) are examined in detail by a group to
detect faults, violations of development standards, and other problems
  – A structured peer review requiring planning, advance preparation, and
    possible rework and follow-up
  – A static test of the software

• Purpose:
  – Prevent the propagation of defects into the operational product
  – Key factor in root-cause analysis to improve process
  – Process control metric

* The term Software Inspection has remained the common name for this type of Formal Review.
Inspections

• Created in 1972 at IBM - Michael Fagan
  – Institutionalized by large software development organizations (e.g., IBM, HP, AT&T)

• Process Improvement input
  – When/Why did error occur that created the defect?
  – How can this be prevented in the future?

• THE Process Control Mechanism for software
  – An aid to productivity as well as quality
  – Finding and fixing defects as they occur instead of trying to test them out
Comparison of Defect Identification Techniques

<table>
<thead>
<tr>
<th>Method</th>
<th>Cost</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspections</td>
<td>Medium-High</td>
<td>&gt;60%</td>
</tr>
<tr>
<td>Walkthrough</td>
<td>Medium</td>
<td>&lt;50%</td>
</tr>
<tr>
<td>Peer Review</td>
<td>Low</td>
<td>&lt;35%</td>
</tr>
<tr>
<td>Self Checking</td>
<td>Low</td>
<td>&lt;20%</td>
</tr>
</tbody>
</table>

Source: Capers Jones, Software Measurement and Estimation

Why?

- More formality and rigor
- Defined methodology for inspections
- Records carefully kept
- All participants are active and responsible
- Repeatable - repeat - Repeatable!
Software Inspections

• CONS
  – MISTAKEN AS A "FINAL INSPECTION" IN THE DEMING SENSE
  – Can add 5-15% to net resources up front*
  – Requires some training
  – Mistaken as too "low tech" to be so effective

• PROS
  – High return on invested time and effort
  – Feedback to developers - avoid injecting defects in future work
  – Serves as checkpoints to facilitate process management
  – Measure performance of tools and techniques
  – Training aid to new or lesser capable people

* About the same amount of time is spent preparing for the inspection as the inspection itself — Rates are approximately one hour per 10 document pages, or one hour per 100 lines of non-commented, non-blank source code.
Benefits of Inspections

- **Defect Reduction**
  50 - 90% of all defects discovered by inspection

- **Cost Improvement**
  10-25% reduction in development costs
  95% reduction in corrective maintenance costs

- **Staff-Hours**
  Overall reduction by 10-40%
  Shortens tail end of schedule

Myers, Computer, August 1990
Inspection Experience Summary

- Inspections are three times more effective than other methods
- Save approximately $1600 for every defect before test (Cost to fix later - cost to find & fix in inspection: $1700 - $105)
- Average inspection discovers 16 defects (4 major, 12 minor) for $25,000 savings
- Some defects cost as much as $10,000 each to fix later

JPL* Experience

Fagan '86

* Jet Propulsion Laboratory

IEEE, Experience Report, August 1990

The difference in the area under these two curves after they intersect represents the "Final Inspection" to detect all the defects introduced during a development process in which the software inspection technique is not used.
Inspections

• Participants have defined roles

  Moderator plans agenda, determines adequate preparation, run meeting, set
tone and pace, act as inspector

  Author provides clarification, act as inspector (usually the best)

  Reader paraphrases line-by-line content, act as inspector

  Recorder records problem and problem classification, acts as inspector, can be moderator

  Inspectors provides expertise in related matters; representatives of previous and subsequent phases

• Inspections have defined steps

  1. Planning  4. Inspection Meeting
  2. Overview (if necessary) 5. Rework
  3. Preparation 6. Follow-up
What Can You Do?

• Do not attend!
  Inspections are not a place for managers

• Ask for a Summary Report

• REJOICE when lots of defects are discovered in inspections!
  Don't use inspection results as a basis for merit or compensation review - Use test results

• Encourage your staff
Reviews and Inspections Summary

- Reviews mark transition/closure
- Software Inspection is the recommended Total Quality Management technique for software process control
- Questions you should ask

What reviews are planned? - Which ones should I attend?
How much of the requirements specification (design, code, test plan) has been (will be) inspected? - May I see the Inspection Management Report?

How have we insured the requirements specification (design) is correct (complete)?
Reviews and Inspections Summary

Action Plan

- Determine what kind of reviews are currently being conducted to control software quality
  
  Discuss with software manager - This can be done this afternoon!

- Initiate an evaluation of software inspections
  
  - Have software manager schedule software developers (and manager) for training in software inspections
    
    Classes taught regionally (east & west coast) and in-house
  
  - Apply inspection to a current work product
    
    Two immediate benefits:
    
    (1) Improved work product
    
    (2) Insight into necessary improvements in development activities leading up to current work product
Configuration Management
Session Goals

• Introduce you to elements of the Software Configuration Management process

• Help you understand that Software Configuration Management is an essential aspect of the software development process
Software Configuration Management (SCM)

• Identification
  – Label, number, and catalog procedures

• Control
  – Product change authority and control procedures
  – Software Configuration Control Board

• Status Accounting
  – Product change recording and reporting

• Audits
  – Physical and functional compliance with requirements prior to release

Only 28% of SEMI/SEMATECH Companies always use Configuration Management
Benefits of SCM

• Control and manage software throughout its life cycle
• Reduce effort to identify and implement changes
• Ensure only necessary changes will be made
• Extend the useful life of software products
• Reduce defects in software releases
• Improve customer satisfaction
• Provide a natural feedback mechanism
Some Common Questions for SCM

• Which software pieces/parts do we control?
  – Internally developed plans, specifications, source, objects, tests, data, environment
  – Purchased software documentation and code

• When does Software Configuration Management begin?
  – During exploration - Sometimes
    Planning, informal control
  – During development - Always
    Planning, personal & development libraries* - control boards
  – During support
    Product or release libraries - release control

* Libraries are defined in attached paper
Some Common Questions for SCM
(Continued)

• How do we control change/release of the software?
  Baseline* control points - Software Configuration Control Board -
  Change log mechanism - Software library-management tools

• What do we report about changes?
  What changes were made - Who approved - Who made change -
  When change was made - How much time was required

• How do we check for completeness, correctness?
  Audit - Before test, before release

* Baseline - The documented identification of a product (with a known set of elements) after some period of development
Configuration Management Summary

- **Software Configuration Management**
  - Is an important management process
  - Is more than version control
  - Can be tailored to the product line or project

- **Questions you should ask**
  - Do we have a Software Configuration Management Plan?
  - Can we rebuild the current release (baseline)?
  - Can we rebuild the previous release (baseline)?

- **Action Plan**
  - Ask for an overview of how problem reporting, and requests for new features, are received, reviewed, and software changes incorporated in your company
Overview

SEMI/SEMATECH companies should adopt a more formal software configuration management (SCM) methodology to manage change activity and control the system software configuration. Part of the methodology should be to adopt a time-phased version (baseline) release process under the more formal SCM approach. (A glossary is attached that provides some common definitions of SCM terms.)

The general benefits to SEMI/SEMATECH companies of a more formal SCM approach include an increased confidence in the ability to preserve the integrity of their software configuration; an ability to provide complete identification of prior, current release, and developmental baselines; and a ready accounting of the status of changes completed, in progress and awaiting work at scheduled times. The benefits to developers include a more stable development environment. A defined set of changes can be integrated as a common development objective and adequate testing can be completed to validate the reliability of the system. The benefits to management include insight into the software engineering process. This allows management to effectively address the long term quality of the equipment software while attending to customer priorities and other market considerations. The benefits to the customer include more accurate information as to when corrections to problems can be expected or when new features will be available.

Structure

An outline of a formal SCM approach is shown in Figure 1. Highlights of this approach include (1) establishment of a six month normal release cycle; (2) an automated change request log based on the current problem report format; (3) establishment of an automated change control mechanism to include a hierarchy of SCM Libraries (Personal or Developmental, Project or Integration, and Release) for control of requirements, design information, code, and test documentation; and (4) establishment of a change control authority (Software Configuration Control Board or SCCB) consisting of representatives from the subsystem groups.
Establishment of a release cycle timeframe is essential to the overall success of maintaining or improving software quality and reliability. Every release of the software, even a “patch,” incurs a level of overhead not only in preparing and shipping the release, but in other activities, such as integration and system testing. Normally, each change has some individual testing performed against it. However, if a release is made for a small number of changes, and the releases are frequent, the system (and regression) testing to support each release becomes a significance portion of the overall support cost. By maintaining a scheduled release cycle, integration and system (regression) testing of the combined release package reduces these overhead costs while significantly increasing the confidence in the delivered software. The increased confidence is attributable to being able to test more thoroughly since (1) each system test can directly or indirectly exercise more of the changed software (many times, the same test is performed to activate two or more components of the software in a sequential process), and (2) there is time to perform more tests because such activities as test setup can be combined. Note that a time-phased release cycle does not mean that no work is performed on the next release until the previous release is shipped. There is usually some overlap if resources will allow it. A six-month release cycle could actually produce releases five months apart as shown in Figure 2.
An automated change request log is necessary to accurately keep track of change requests and reduce the burden on personnel to manually mark, retrieve, sort, store, etc., the change requests in hardcopy form.

Establishment of a hierarchy of libraries provides a more secure mechanism to maintain the integrity of released software through promotion from one library to the next. Promotion is a control mechanism that requires changed software to be subjected to an established level of scrutiny before it is formally approved for entry into a higher level library. (The SCCB or an inspection team acting for the SCCB may promote software components). This control, coupled with the ability to manage the configuration of the documentation defining the software system, reduces the defects because of incorrect or outdated support documentation, and increases the potential to reuse significant portions of the software when developing the next generation equipment.

The change control authority would normally consist of a few software developers, system engineers, and test/quality engineers (if they exist). These would be personnel most capable of rendering accurate decisions about the mix of change activity that should be attempted for any given release cycle. Completing this decision process sometimes forces issues to be addressed and resolved that would perhaps go unresolved (or even unnoticed) until the consequences become unrecoverable. Customer requests (demands) for new features may or may not be more important than the correction of defects affecting operational uptime and throughput.

**Change Request/Problem Reporting**

All change activity is processed through a Log function. A Librarian would log the change request. The change request could be for a problem report (e.g., field failure report) a request for a new feature or a request for a conversion effort to accommodate new hardware or software (e.g., extensive update to operating system impacting application programs). Table 1 contains a summary of change types appropriate for software.
Table 1  Change Profiles

<table>
<thead>
<tr>
<th>Principle Input Source(s)</th>
<th>Correction</th>
<th>New Feature</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Report, Test Failures</td>
<td>Customer Request, SEMATECH Initiative, Internal Initiative</td>
<td>Internal Initiative- (Hardware Technology Driven), Vendor Driven - (Hardware or Support Software)</td>
<td></td>
</tr>
<tr>
<td>Origin</td>
<td>Requirement Not Correct, Non-conformance to Requirements (Design, Code Defect) - (Bug or Possible Bug)</td>
<td>New/Changed Requirement - (Suggestion, New Item, or Improvement)</td>
<td>New/Changed Requirement - (New Item or Improvement)</td>
</tr>
<tr>
<td>Priority Level</td>
<td>Normal, Important (Urgent), or Critical (Emergency)</td>
<td>Usually Normal or Important (Urgent)</td>
<td>Usually Normal</td>
</tr>
<tr>
<td>Complexity</td>
<td>Low, Medium, High</td>
<td>Low, Medium, High</td>
<td>Low, Medium, High</td>
</tr>
</tbody>
</table>

The change request/problem report log becomes the basis for tracking status (open, closed, deferred, or rejected) and implementation in releases.

**Change Control/Configuration Management**

The activities of change control would be performed by a group commonly referred to as the SCCB. The SCCB should have a Chairperson, a Librarian (maybe one for each engineering group or a single person if dedicated to Librarian functions), and sufficient representation and decision authority to determine the change activity for each release cycle. This may require coordination with other groups within the company or within the development project, with individual customers, and with SEMATECH. The coordination would establish that features/customer requests and initiatives can be accommodated within a given release cycle along with corrections needed based on field reports and internal test failures. The SCCB should plan to meet on a regular basis, and whenever emergency requests must be processed.

Preliminary analysis (Figure 1) is accomplished to provide the SCCB with sufficient information to make decisions quickly as to accept, defer, or reject changes. The information necessary for preliminary analysis is highlighted in Table 2. It is essential that the SCCB not get sidetracked on troubleshooting or design issues at this point in the process. The purpose is to weigh the analysis information against the resources available and establish a status for the change. Depending on the nature of the change, the board may have to suspend activity on one or more changes to accommodate an extremely sensitive change. The value of the preliminary analysis in this situation is to provide enough insight into that sensitive change to determine if it is even possible and really prudent to take such action at that time. A decision to reject a change is usually accompanied by the reasoning used to make that decision, which could be that insufficient information precluded the ability to analyze the problem. A decision to defer is usually based on insufficient analysis and the change request is returned to the analyst for more information. A decision to accept is accompanied by a decision as to which release the change should be scheduled for. The change implementation package is then sent to the developer for implementation. The decision is also communicated back to the originator.
Table 2  Contents Of Change Request Analysis

<table>
<thead>
<tr>
<th>Correction</th>
<th>New Feature</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated Time:</td>
<td>Estimated Time</td>
<td>Estimated Time</td>
</tr>
<tr>
<td>(Person-Months)</td>
<td>Estimated Time</td>
<td>Estimated Time</td>
</tr>
<tr>
<td>Possible causes/Most likely</td>
<td>Sizing/Timing Estimate -</td>
<td>Relevant impact of new hardware -</td>
</tr>
<tr>
<td></td>
<td>Addition computational resources</td>
<td>new operating system/application</td>
</tr>
<tr>
<td></td>
<td>required?</td>
<td></td>
</tr>
<tr>
<td>Hardware/Software interface impact?</td>
<td>Hardware/Software interface</td>
<td>Hardware/Software interface</td>
</tr>
<tr>
<td></td>
<td>impact?</td>
<td>impact?</td>
</tr>
<tr>
<td>Software only to correct?</td>
<td>Test Outline</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modify existing tests? - New tests required?</td>
<td>Modify existing tests? - New tests required?</td>
</tr>
<tr>
<td></td>
<td>Changes required to User information?</td>
<td>New User information required?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New User information required?</td>
</tr>
</tbody>
</table>

The mechanism to handle a safety-related (personnel or equipment) critical or "emergency" change (perhaps even a market-strategic change) is to send the change directly into development, apply all available resources to make the change, test it (usually limited testing), and issue an "interim release." Changes of this nature are usually folded back into the normal SCM process for a thorough review and complete testing before inclusion into the next normal version release.

The development activity proceeds through testing of the change and completion of documentation. As soon as practical, user information is passed to any documentation group such as a Technical Communications group for inclusion into user publications and feedback to the originator or a representative user group. The SCCB evaluates the results of each completed change package (Table 3) for promotion into the Release Library. The change package itself can consist of the output and reporting of very structured verification and validation activities depending on the nature of the change. The entire SCCB or a portion of it may function in this evaluation activity. Promotion from the Personal Library to the Project (Integration) Library may be the result of an inspection of the change. Small groups (3 to 5 people) have been shown to very effective at discovering defects prior to test by using the inspection technique. Acting as the change control authority, inspection teams form to approve changes (singularly or in groups) for promotion to the Project Library for integration and system testing.

Table 3  Contents of Change Implementation Package

<table>
<thead>
<tr>
<th>Correction</th>
<th>New Feature</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Time</td>
<td>Actual Time</td>
<td>Actual Time</td>
</tr>
<tr>
<td>(Person-Months)</td>
<td>Actual Time</td>
<td>Actual Time</td>
</tr>
<tr>
<td>Old &amp; New files or</td>
<td>Requirements and Design</td>
<td>Old &amp; New files or New file and</td>
</tr>
<tr>
<td>&quot;Diff List&quot;</td>
<td>information - Source Code</td>
<td>&quot;Diff List&quot;</td>
</tr>
<tr>
<td></td>
<td>Inspection Report</td>
<td>Inspection Report</td>
</tr>
<tr>
<td></td>
<td>Peer Review comments and/or Walkthrough results</td>
<td>Inspection Report and/or Walkthrough Results</td>
</tr>
<tr>
<td></td>
<td>Inspection Report</td>
<td>Inspection Report</td>
</tr>
<tr>
<td></td>
<td>Test Cases - Results</td>
<td>Tests completed - New Test Cases - Results</td>
</tr>
<tr>
<td></td>
<td>Updated documentation</td>
<td>Updated documentation</td>
</tr>
</tbody>
</table>
Prior to final release, the SCCB insures sufficient system testing has been completed (either in-house or at a Beta site) to establish the overall integrity and reliability of the version to be released.

Note the metric of estimated time (Table 2) and actual time (Table 3). This information and an estimate of the complexity of the change (Table 1) could provide the basis of a longer term effort to more accurately estimate the resources required to perform support activities.

**Software Engineering Activities**

The software engineering activities center on analysis, development and some of the promotion methods. Analysis is performed by a person knowledgeable in the area of the change request. The software developer benefits from this exercise because it provides him or her a preliminary look at the request and the opportunity to begin thinking about the problem without the pressure to immediately start solving it.

Development is of course the application of the same engineering activities in the support environment that occurred in the original development. Taken as a group, the approved changes for any particular release are in fact, a whole new development effort. Part of the engineering activity should include software inspections also mentioned as part of the change control activity. Software inspections are an engineering activity that serves management issues as well. Because software inspections are performed on completed products (at least a minimum number of TBDs in the case of requirements), they herald milestones and transitions as well as being the most efficient defect removal mechanism in the software developer's inventory.

The SCM Library hierarchy must not be confused with other libraries that are used to group similar software entities such as support tools, scripts, include files, test files, etc. Some of these libraries may be found at multiple levels of the SCM Library structure depending on whether they have files being modified as part of a change package.

**Summary**

The proposed Software Configuration Management philosophy and Time-Phased version release process are not readily separable. They are both key components to improving the quality and reliability of equipment software.

As mentioned earlier, a software package specifically designed to automate much of the report tracking and control of the SCM Libraries is important to the success of this concept.
13 GLOSSARY OF SOFTWARE CONFIGURATION MANAGEMENT TERMINOLOGY

Baseline: A baseline is the documented identification of a software product CI -- its code and all its related documentation at some specific point in time. It is the basis for all SCM activities.

CCB: Configuration Control Board. A Configuration Control Board is a group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. A CCB is sometimes referred to as a Change Control Board.

CI: Configuration Item. A collection of (software) elements (or components) treated as a unit for the purpose of configuration management. Configuration Items can vary in size, complexity, and type. A CI may also be called a Computer Software CI (CSCI) or Computer Program CI (CPCI).

CM: Configuration Management.

CPC: Computer Program Component, same as CSC.

CSC: Computer Software Component. A lower level entity of a CI. A CSC may be the requirements or a collection of the design information for a CSCI.

Inspection: (1) A formal evaluation technique in which software requirements, design, or code are examined in detail by a group to detect faults, violations of development standards, and other problems. (2) A structured peer review requiring advance preparation and planning. (3) Software inspections are formal peer reviews of code or associated documentation including requirements specifications, design descriptions, and test plans. Any product generated during the course of the software life-cycle can be inspected, although it may not always be advantageous to do so. Software inspections are formal in that there is a set agenda. A limited amount of material is covered in a set amount of time. Those attending the review have clearly defined roles. All issues raised at the inspection are rigorously recorded and resolved. Software inspections are peer reviews in that management is generally not invited to attend.

Personal Library: The Personal Library (also termed the dynamic or programmer's library) is used for holding newly created or modified software entities. This library constitutes a software developer's workspace for writing new code or documentation, and may take any form suitable to the developer's needs, but should have a degree of order to it that allows the status of its entities to be determined easily. Each software developer may have a Personal Library from which project entities can be linked and/or copied. Or, each small team may have a Personal Library assigned to the team with lower level personal libraries assigned to each individual, especially in a local area network (LAN) or mainframe environment. Access to the Personal Library is controlled by, and usually limited to, the software developer.
**Project Library:** The Project Library (also termed the controlled or "master" library) is a library used for managing the current internal developmental baselines and for controlling changes made to them. This library represents the latest internally-approved version of the software product being developed. Changes to software entities in the Project Library should have gone through formal approval procedures established in a configuration management plan. Code in this library should have been tested sufficiently to assure that it is ready for integration. Copies may be freely made for use by the software developers and others, but changes must be strictly controlled and documented in order to ascertain at any given point its exact configuration. Even for simple projects in which there is only one code developer, there still should be functional separation between the Personal Library and the Project Library.

**Promotion:** A promotion is an action taken with a software component to increase the level of authority needed to approve changes to it. For example, a top-level software design description is promoted or moved into the Project Library where all developers can view, but not modify it without proper authority. This allows the developers to work on issues that may concern detailed designs and implementation.

**Release:** A release is a copy of the software CI that is turned over to the customer or user. It is a promotion of that CI outside of the development organization.

**Release Library:** The Release Library (also termed the static or "software repository" library) is a library used to archive the various baselines released for general use. This library is never changed (except to add a new release baseline), since it must be able to duplicate results from software that has been released for operational use by other organizations. Access should be limited to "read only" for the purpose of making copies.

**Revision:** A revision is a formal change to a software CI that does not alter its documented functional or performance capabilities. An example of this is when code is changed to correct a fault.

**SCCB:** Software Configuration Control Board. A Software Configuration Control Board is a group of individuals who oversee the software change process, with ultimate authority for approving a change and promoting a software unit from one library to another. The individuals may be from the project, related organizations and management levels, the customer, or some combination. During the development process, the SCCB controls promotions into the Project Library from the Personal Library and changes to the products in the Project Library. During the support phase, the project CCB and SCCB provide the authority to make changes to products already promoted to the Release Library, and to promote software products from the Project Library to the Release Library. May also be referred to as a Software Change Control Board. The SCCB and the CCB may be the same in many applications.

**SCM:** Software Configuration Management.

**SCMP:** Software Configuration Management Plan.
**Software Component:** A distinct part of a Software Configuration Item (CI). A software component may also be called a Computer Software Component (CSC) or Computer Program Component (CPC). CSCs may also be further decomposed into other CSCs or individual units. If a large analysis program is called out as a CI, some of the CSCs could be a requirements document or major groupings of the software modules.

**Version:** A version is a software CI with a defined set of capabilities. A new version is a variation of the previous version, in that it has a change in its functionality or performance characteristics. An example of this is if the system is changed to generate a different output.
Session Goals

• To understand the purpose of software testing

• To introduce testing approaches and activities
Basic Goal of Software Testing

The basic goal of software testing is the elimination of defects in the software product. An additional benefit is that we show that it works in the operational environment.
A Testing Philosophy

- Testing is the act of performing test (implies formality)
- Software requirements are essential to testing
- Testing is predominately an effort to show that the software does not meet its specification
  Emphasis on "Negative Testing" - But there is an additional benefit!
- Effective testing detects defects
- Defects can occur at many stages
  - Requirements Specification  C  Design  C  Implementation  C  
    Integration  C  Support
  - Must test at each step (reviews/inspections are also test activities)!
A Test Philosophy
(Continued)

• Testing to detect all the defects is much like using "Final Inspection" to separate good product from bad

• Regression testing should be used for change packages and critical or major changes

• Software cannot be completely tested—Select subset of all possible test cases that have the highest probability of detecting the most defects

  Functional (Feature) - Structural - Performance and Stress; combined carefully to perform Reliability testing
Testing Approaches

• Static Testing
  – Testing activities conducted during software development that do not involve the execution of the software - Static analyzers, Inspections

• Dynamic Testing
  – Testing processes that are performed on executable or executing software
  – "White Box" Testing: Requires knowledge of code structure - Implements logic, statement, and path coverage
  – "Black Box" Testing: Clear, complete specifications necessary - Knowledge of program structure not necessary
  – Both require:
    Ingenuity - Intuition - Experience
    Knowledge of previous defects and weaknesses
Developing / Recording Tests

- Test Plan
  Only 8% of SEMI/SEMATECH Companies always produce a Test Plan - 15% never do

- Test Design

- Test Case Specification

- Test Procedure Specification

- Test Records
  Only 3% of SEMI/SEMATECH Companies always maintain a test log - 20% never do
Testing Summary

- Testing should detect defects; proper functioning is an additional benefit
- Requirements are critical
- Adequate testing requires planning
- Test documentation is a "must" - Regression testing
- Questions you should ask
  - Do we have a test plan? - Are tests defined?
  - Have we saved the results?
  - Can we rerun the tests and compare with previous results?
Testing Summary
Action Plan

• Review the schedule for testing with your software manager
  - Ask for an optimal estimate to adequately test
  - Is the difference too uncomfortable?

• Ask for an overview of the test coverage and the trends of the rate of failures discovered and defects corrected
  - Is the rate of failures discovered greater or less than the rate of defects corrected?
  - Does this make you feel good about releasing the product?

• Seriously consider the appropriateness of the target release date given these two sets of inputs
Support (Review and Discussion)
Building Reliability Into Software

• Define/Understand the Process
  The framework in which development occurs

• Adopt a Development Methodology
  – Not whether we'll have a methodology, but how rigorous will it be
  – Investment in software engineering methods (Tools - Training) will provide high return on investment

• Develop Clear, Unambiguous Requirements
  Common faults: Omitting requirements needed to satisfy customer needs; untraceable requirements; complex writing styles
Building Reliability Into Software
Continued

• Use Software Inspections (Formal In-Process Reviews)
  – A "Value Added" activity - Reduces total cost (development and support), test time (without sacrificing coverage), and defects in shipped software - Part of the investment in software engineering methods
  – Key input to process improvement activities
  – Allows checking - On Track? - On Schedule?

• Apply Configuration Management
  – Change control: Automate
  – Part of the investment in software engineering methods

• Plan for Testing
  – All testing is important: Functional (Feature), Structural, Performance and Stress: é Reliability . . . and Regression
  – Is there sufficient time in the schedule? Or is it being squeezed?
Building Reliability Into Software
Concluded

• Measure the Process
  – Planned vs Actual: Size, Cost, People, Schedule
  – Defects removed by phase (Requirements, Design, Coding) vs failures
during test (caused by defects) vs field reports (more defects)

• Perform Root-Cause Analysis
  – What type of fault; when, why introduced; when found; why not
detected; what could have been done to prevent

• Follow-up with Audits and Lessons Learned
  – Review existing practices - headed in the right direction?
  – Analysis of customer feedback (need to ask for it!)
  – What worked and what didn't
Where is YOUR Organization?

Software Engineering Capability

**Maturity Level**

5 - Optimizing
   [Feedback]

4 - Managed
   [Quantitative]

3 - Defined
   [Qualitative]

2 - Repeatable
   [Intuitive]

1 - Initial
   [Ad Hoc]

**Results**

**Acquired Capabilities**
- Adaptable to Change
- Automation

**Process Control**
- Statistical Process Control
- Changing Technology
- Problem Prevention

**Process Measurement**
- Fault Projection
- Test & Review Coverage
- Process Metrics Database

**Process Definition**
- Quality Point Reviews
- Software Engineering Training

**Basic Management Control**
- Project Planning (Cost, Schedule)
- Configuration Management
- Regression Testing
- Software Quality Assurance

**Software Process Maturity Distribution**

62% 87%

SEI-assisted: 13 sites, 63 Projects
Self-assessment: 46 sites, 233 Projects

Your Role in Software Development

- Understand the discipline and controls of software engineering

- Ask the RIGHT questions

  The last two:

  - How do we plan to maintain this software?
    (This question should be asked early in the project!)
  - How can I help?

- Encourage

  - Motivation to do it right comes from the top!
  - Quality improvement starts with you
Summary

• High quality, reliable software doesn't just happen

• High quality software requires sufficient front-end loading—Training & Tools

• Two high value-added activities—Requirements & Inspections

• Adherence to process reduces risk of poor software

• Improving the process begins with management
Summary
Action Plan

Investigate the maturity of your company's software development process capability

− Ask your software manager (lead engineer) to informally apply the SEI Level Two Software Maturity Survey questions to your software development process and report on the results
− Stress being frank and open
− Together, target one or two areas that will improve your company's ability to meet level two attributes
− Re-assess this area in 4 to 6 months
− Target one or two more . . . . . .
16 QUESTIONS TO ASK

1. Have we really defined OUR process?
2. What is our policy for software quality?
3. Do we have standards for software development? - What are they?
4. Do we have a Software Development Plan? - Is it current?
5. Do we have a cost and schedule estimate?
6. How are we tracking to the development schedule?
7. Do we know what the requirements are?
8. What analysis techniques are we using to define/refine requirements?
9. Are the requirements documented?
10. Who is our customer? - Who is the user?
11. Have we drafted the System Test Plan?
12. What reviews are planned? - Which ones should I attend?
13. How have we insured the requirements specification (design) is correct (complete)?
14. How much of the requirements specification (design, code, test plan) has been (will be) inspected?
15. May I see the Inspection Management Report?
16. Do we have a Software Configuration Management Plan?
17. Can we rebuild the current release (baseline)?
18. Can we rebuild the previous release (baseline)?
19. Do we have a test plan? - Are tests defined?
20. Have we saved the results?
21. Can we rerun the tests and compare with previous results?
22. How do we plan to maintain this software?
23. HOW CAN I HELP?
17 ACTION PLAN FOR SOFTWARE QUALITY

1. Make an appointment with your software manager to sketch out the structure of YOUR software development process
   Use blank Embedded Life Cycle Model as basis or develop simpler model
   Compare to model in this course
   Ask: Are there elements of the course model that we want to implement? If so: When can we begin to implement them? What can we do in the next 6 months? In the next year?

2a. Write out a policy statement defining YOUR company's concept of quality for software

   Sample Policy Statement
   “Quality software shall be an objective of this company's quality program. This objective applies to the acquisition, development, use, and support of software. The program shall address, as a minimum:
   
   Organization, tasks, and responsibilities; software documentation; configuration management; testing, and; reviews and audits.”

2b. Make an appointment with your software manager to outline software development goals for your next new feature (or next generation of equipment)
   Integrate Process Model from previous action item - Technical risks - Analysis and Design methods - Initiate formal schedule and budget estimation, define work packages
   Use as basis for a more formal Software Development Plan

3a. Meet with (interview) customer production managers
   Ask: What are their top five requirements for your company's equipment.

3b. Meet with your system/project and software managers
   Ask: How are we meeting these requirements?

4a. Determine what kind of reviews are currently being conducted to control software quality
   Discuss with software manager - This can be done this afternoon!

4b. Initiate an evaluation of software inspections
   • Have software manager schedule software developers (and manager) for training in software inspections
     Classes taught regionally (east and west coast) and in-house.
   • Apply inspection to a current work product
     Two immediate benefits: Improved work product - Insight into necessary improvements in development activities leading up to current work product
5. Ask for an overview of how problem reporting, and requests for new features, are received, reviewed, and software changes incorporated in your company.

6a. Review the schedule for testing with your software manager
   Ask for an optimal estimate to adequately test
   Is the difference too uncomfortable?

6b. Ask for an overview of the test coverage and the trends of the rate of failures discovered and defects corrected
   Is the rate of failures discovered greater or less than the rate of defects corrected?
   Does this make you feel good about releasing the product?

6c. Seriously consider the appropriateness of the target release date given these two sets of inputs

7. Investigate the maturity of your company's software development process capability
   Ask your software manager (lead engineer) to informally apply the SEI Survey Level Two questions to your software development process and report on the results
   Stress being frank and open
   Target one or two areas to improve your company's ability to meet level 2 attributes

Re-assess this area in 4 to 6 months . . . . . Target one or two more
18 SEI MATURITY MODEL: LEVEL 2 – REPEATABLE PROCESS

At Maturity Level 2, the organization uses standard methods and practices for managing software development activities such as cost estimating, scheduling, requirements changes, code changes, and status reviews. An organization qualifies at Level 2 if the percentage of affirmative answers is at least 80% and the percentage of affirmative answers to shaded questions is at least 90%. (The answers to these questions should reflect standard organizational practices)

SEI defines mechanism and formal procedure as follows:

- Mechanism—a means whereby the performance of a task, procedure, or process is assured.
- Formal Procedure—a documented series of steps with guidelines for use.
<table>
<thead>
<tr>
<th>Ques. No.</th>
<th>SEI Question – Level 2</th>
<th>Proj 1</th>
<th>Proj 2</th>
<th>Proj 3</th>
<th>Notes (Follow-up Ques)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1</td>
<td>For each project involving software development, is there a designated software manager?</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.1.2</td>
<td>Does the project software manager report directly to the project (or project development) manager?</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.1.3</td>
<td>Does the Software Quality Assurance (SQA) function have a management reporting channel separate from the software development project management?</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.1.6</td>
<td>Is there a software configuration control function for each project that involves software development?</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Is there a required training program for all newly appointed development managers designed to familiarize them with software project management?</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>1.3.1</td>
<td>Is a mechanism used for maintaining awareness of the state-of-the-art in software engineering technology?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Is a formal procedure used in the management review of each software development prior to making contractual commitments?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Is a formal procedure used to assure periodic management review of the status of each software development project?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Is there a mechanism for assuring that software subcontractors, if any, follow a disciplined software development process?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1.7</td>
<td>For each project, are independent audits conducted for each step of the software development process?</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2.1.9</td>
<td>Are coding standards applied to each software development project?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1.14</td>
<td>Is a formal procedure used to make estimates of software size?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1.15</td>
<td>Is a formal procedure used to produce software development schedules?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1.16</td>
<td>Are formal procedures applied to estimating software development cost?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.1.17</td>
<td>Is a mechanism used for ensuring that the software design teams understand each software requirement?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Are software staffing profiles maintained of actual staffing versus planned staffing?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Ques. No.</td>
<td>SEI Question – Level 2</td>
<td>Proj 1</td>
<td>Proj 2</td>
<td>Proj 3</td>
<td>Notes (Follow-up Ques)</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>------------------------</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Are profiles of software size maintained for each software configuration item, over time?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Are statistics on software code and test errors gathered?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Are profiles maintained of actual versus planned software units designed, over time?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.8</td>
<td>Are profiles maintained of actual versus planned software units completing unit testing, over time?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.9</td>
<td>Are profiles maintained of actual versus planned software units integrated, over time?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.10</td>
<td>Are target computer memory utilization estimates and actuals tracked?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.11</td>
<td>Are target computer throughput utilization estimates and actuals tracked?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.12</td>
<td>Is target computer I/O channel utilization tracked?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.16</td>
<td>Are software trouble reports resulting from testing tracked to closure?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.18</td>
<td>Is test progress tracked by deliverable software component and compared to the plan?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.2.19</td>
<td>Are profiles maintained of software build/release content versus time?</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Does senior management have a mechanism for the regular review of the status of software development projects?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.4.5</td>
<td>Is a mechanism used for regular technical interchanges with the customer?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.4.7</td>
<td>Do software development first-line managers sign off on their schedules and cost estimates?</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2.4.9</td>
<td>Is a mechanism used for controlling changes to the software requirements?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.4.17</td>
<td>Is a mechanism used for controlling changes to the code? (Who can make changes and under which circumstances?)</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2.4.20</td>
<td>Is there a mechanism for assuring that regression testing is routinely performed?</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
19  FOLLOW-UP QUESTIONS

When performing a self assessment, use the requested information in these follow-up questions as benchmarks to determine if your organization/project should answer Yes to the original question.

1. Where responsibility assignments are questioned, request the name of a specific individual, tenure in job, job description, and evidence of activity, such as monthly reports, meeting reports, control logs.

2. Where the existence of a group is questioned, request names of members, the organization represented, and recent meeting agendas and minutes.

3. Where the existence of education or training programs is questioned, request the schedule of recent courses offered, course outlines, names of attendees, and qualifications of instructors and students.

4. Where the existence of a mechanism, procedure, standard, criteria, or guideline is questioned, request a copy of the controlling document, its revision history, the name of the individual(s) responsible for tracking, job description(s), and recent issue/activity reports.

5. Where the use of profiles, tracking reports, planned vs. actual comparisons, and measurements are questioned, request the three most recent reports, measurement summaries, or comparisons.

6. Where computations or analysis of data is questioned, request copies of the most recent computations, analysis reports, or summaries showing results or conclusions reached.

7. Where the initiation of actions are questioned, request copies of recent action tracking and/or summary reports.

8. Where the conduct of certain actions or use of facilities is questioned, request evidence in the form of procedures, responsibilities, or tracking systems to demonstrate performance.

9. Where the existence of a facility, capability, practice, or method is questioned, request supporting evidence in the form of inventory lists, tracking and usage reports, instruction manuals, education programs, etc.

10. Where the use of an automated tool or facility is questioned, request a demonstration of that tool or facility.
20 SOME FURTHER READING

Articles Specific to Sessions

Introduction

Humphrey, Watts S.

Process Organization

Basili, Victor R., and Musa, John D.

Project/Risk Management

Boehm, Barry W.

Reviews and Inspections

Fagan, Michael E.

On Software Project Management

Brooks, Fred

DeMarco, Tom

Glib, Tom

Utz, Walter J., Jr.

On Software Metrics

Boehm, Barry

DeMarco, Tom
Controlling Software Projects: Management, Measurement, and Estimation, Yourdon Press, 1982

Fenton, N. E.


Grady, R. and D. Caswell


Jones, Capers


Putnam, L. H., and Myers, W.
