A Fundamental Approach to Improving Software Quality

SQGNE 9 February 2011

Rick Spiewak
The MITRE Corporation
rspiewak@mitre.org

What is Software Quality Management?:

An Application of the Basic Principles of Quality Management

"Quality is free. It's not a gift, but it is free. What costs money are the unquality things – all the actions that involve not doing jobs right the first time." [1]

[1] "Quality Is Free: The Art of Making Quality Certain", Philip B. Crosby. McGraw-Hill Companies (January 1, 1979)

What is Software Quality Management?:

An Application of the Basic Principles of Quality Management

"You can't inspect quality into a product." [2]

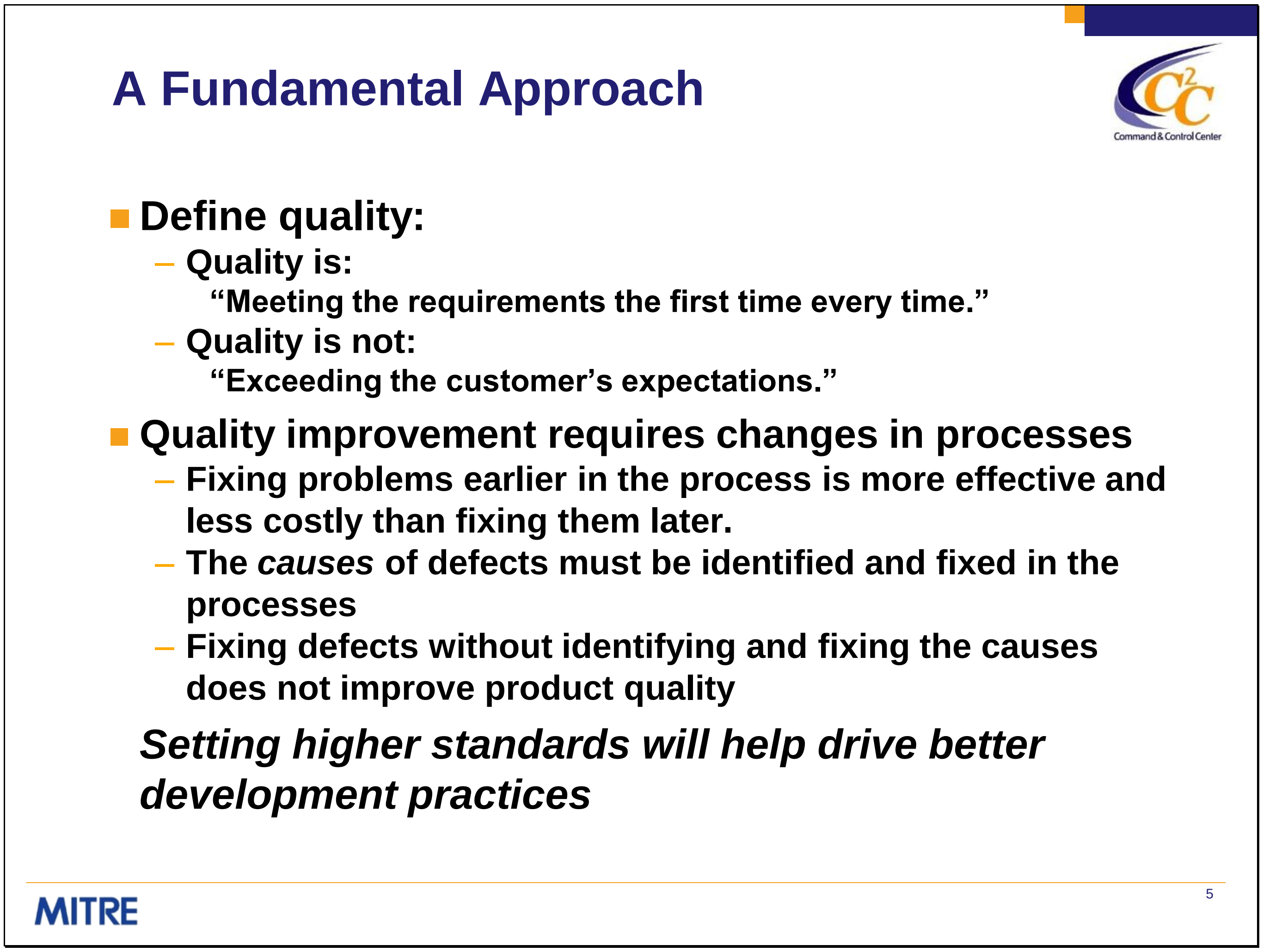
[2] Harold F. Dodge, as quoted in "Out of the Crisis", W. Edwards Deming. MIT, 1982

What is Software Quality Management?:

An Application of the Basic Principles of Quality Management

"Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often." [3]

[3] "Code Complete 2" Steve McConnell. Microsoft Press 2004

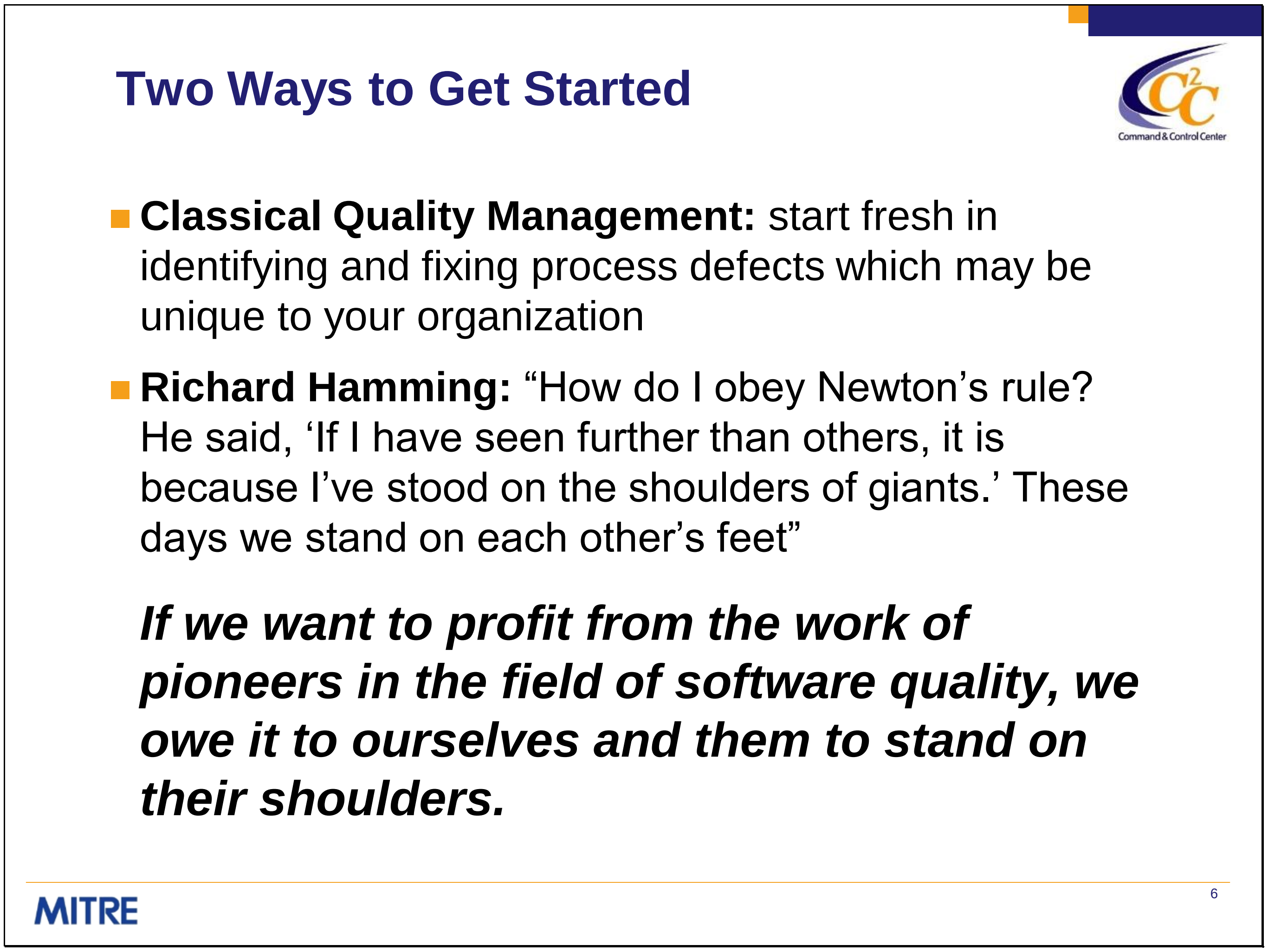


A Fundamental Approach

- **Define quality:**
  - **Quality is:**
    "Meeting the requirements the first time every time."
  - **Quality is not:**
    "Exceeding the customer's expectations."
- **Quality improvement requires changes in processes**
  - **Fixing problems earlier in the process is more effective and less costly than fixing them later.**
  - **The *causes* of defects must be identified and fixed in the processes**
  - **Fixing defects without identifying and fixing the causes does not improve product quality**

Setting higher standards will help drive better development practices

"Meeting the requirements" means knowing what is required and doing just that.
"Exceeding the customer's expectations" is particularly troublesome in software development: it enables feature creep and ignores the extra costs of documentation, training, support and sustainment



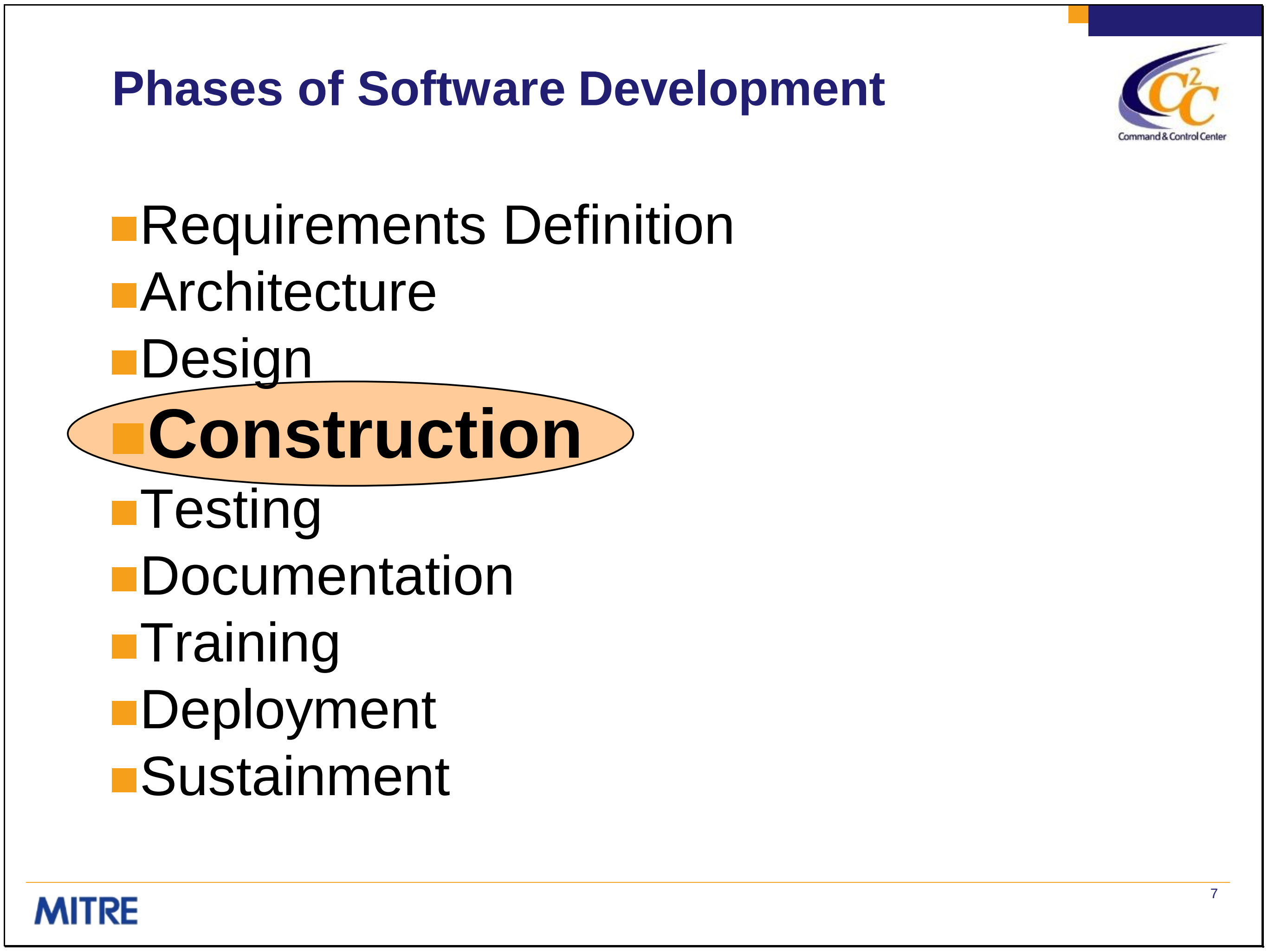
Two Ways to Get Started

- **Classical Quality Management:** start fresh in identifying and fixing process defects which may be unique to your organization
- **Richard Hamming:** “How do I obey Newton’s rule? He said, ‘If I have seen further than others, it is because I’ve stood on the shoulders of giants.’ These days we stand on each other’s feet”

If we want to profit from the work of pioneers in the field of software quality, we owe it to ourselves and them to stand on their shoulders.

Hamming, Richard. You and Your Research. Transcription of the Bell Research Colloquium Seminar, 7 Mar. 1986.

A rigorous approach to Quality Management would start fresh and analyze specific root causes, leading to organization-specific process changes. However, there are well-tested best practices which can be implemented without having to discover them anew in each organization.



Phases of Software Development

- Requirements Definition
- Architecture
- Design
- **Construction**
- Testing
- Documentation
- Training
- Deployment
- Sustainment

MITRE

Just as in the case of other processes, all phases of software development are candidates for quality management.
This discussion is focused on the construction phase
Requirements definition has been addressed only by including quality related requirements. However, this is a likely candidate for major improvements.

What's Wrong With Software Construction?

- Historically a "write-only" exercise:
  If it doesn't break, no one else reads it
- Ad-hoc or absent standards
- Testing as a separate exercise
- Re-work (patch) to fix defects ("bugs")
- Features take precedence over quality
- Definition of quality is not rigorous

Standards and best practices are not uniformly followed because they are not normally stated as requirements

What's Missing in Software Construction?

If we built buildings this way….

They might not stay standing

Or, we might not

Buildings are *not* built this way

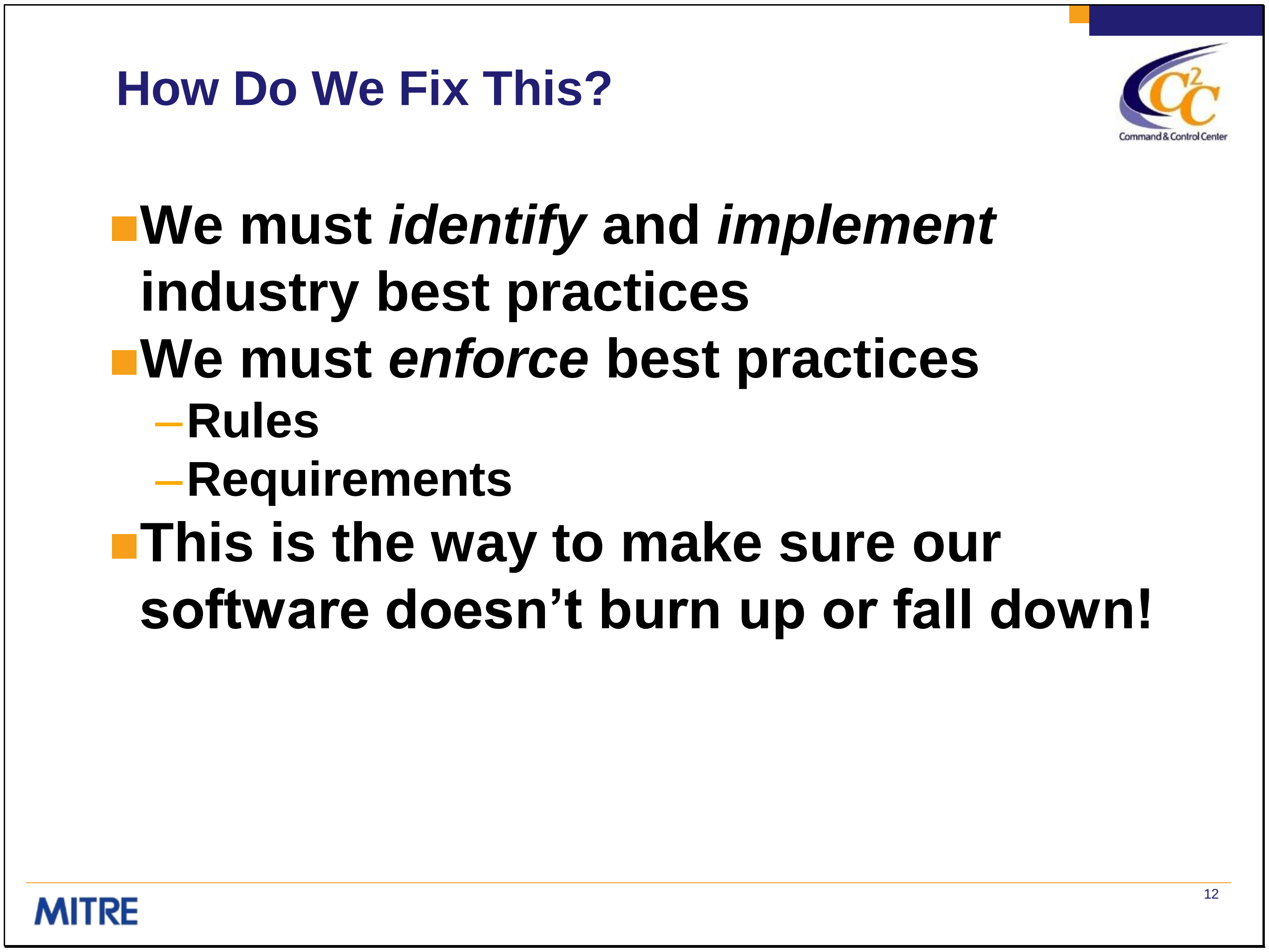
Building construction has standards!

Typical Building Code Requirements:

- Building Heights and Areas
- Types of Construction
- Soils and Foundations
- Fire-Resistance and Fire Protection Systems
- Means of Egress
- Accessibility
- Interior Finishes and Environment
- Energy Efficiency
- Exterior Walls
- Roof Assemblies
- Rooftop Structures
- Structural Design
- Materials (Concrete, Steel, Wood, etc.)
- Electrical, Mechanical, Plumbing….

Missing: the “Building Code” for software

- There is a lack of uniformity and standards
- Historically, these are created *ad hoc* by each organization
- There is no penalty for inadequate standards
- Best practices are often discarded under cost and schedule pressure



How Do We Fix This?

- We must *identify* and *implement* industry best practices
- We must *enforce* best practices
  - Rules
  - Requirements
- This is the way to make sure our software doesn't burn up or fall down!

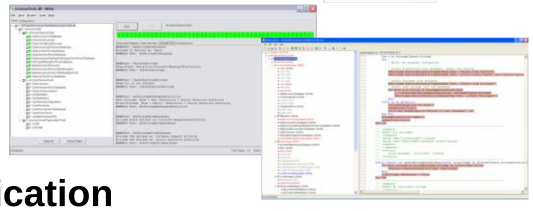
One weakness of a traditional approach needs to be addressed directly. That is, the idea of specifying the results we want, coupled with a reluctance to specify how to achieve them. In the case of known best practices, we need to be specific. If a development organization doesn't already use best practices, they are unlikely to adopt them for our project unless we require it.

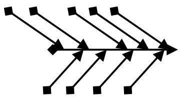
Improving Development Practices:

Best Practices in Software Development

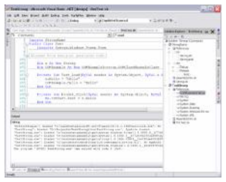
- Uniform Coding Standards
  - References
  - Tools
  - Practices
- Automated Unit Testing
  - Design for test
  - Tools for testing
  - An Enterprise approach
- Root Cause Analysis and Classification
  - Analytic methods
  - Taxonomy
- Code Reuse
  - Development techniques
  - Reliable sources

Framework Design Guidelines
Conventions, Idioms, and Patterns for Reusable .NET Libraries
.net Development Series
Effective Java Programming Language Guide
Joshua Bloch
Practical Guidelines and Best Practices





Top level categories :
- 0xxx Planning
- 1xxx Requirements and Features
- 2xxx Functionality as Implemented
- 3xxx Structural Bugs
- 4xxx Data
- 5xxx Implementation
- 6xxx Integration
- 7xxx Real-Time and Operating System
- 8xxx Test Definition or Execution Bugs
- 9xxx Other



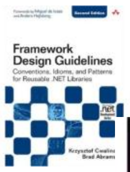
Coding standards are expressed in published books, and checked by tools and peer reviews
We need to make automated unit testing pervasive, with techniques shared by developers and testers.
When developers have to create unit tests, their approach to programming changes in order to make the code more testable.

Analytic methods for Root Cause analysis include “Five Whys”, and Kepner-Trego Problem Analysis.

Code Reuse, when done sensibly, reduces both effort and defects.

Improving Development Practices:
Uniform Coding Standards

- References
  - .NET
    - Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries
    - Practical Guidelines and Best Practices for Microsoft Visual Basic and C# Developers
  - Java
    - Effective Java Programming Language Guide
    - The Elements of Java Style
- Tools and Techniques
  - Static Code Analysis
    - .NET
      - FxCop
      - DevPartner Studio
    - Java
      - FindBugs (Eclipse plug-in)
      - ParaSoft JTest
  - Code Review (with Government audit)



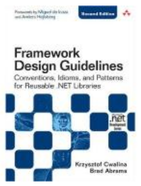
MITRE

Details on references:
Cwalina, Krzysztof and Abrams, Brad, *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries (2nd Edition) (Microsoft Net Development Series)*, Addison-Wesley Professional (2008)
Balena, Francesco and Dimauro, Giuseppe, *Practical Guidelines and Best Practices for Microsoft Visual Basic and C# Developers*
Bloch, Joshua, *Effective Java Programming Language Guide,* Prentice Hall (2001)
Ambler, Scott et al, *The Elements of Java Style*, Cambridge University Press (2000)

Improving Development Practices: Tools

.NET: Enterprise Coding Standards – Static Analysis with FxCop

- Microsoft developed free tool
- Equivalent version in Visual Studio
- Analyzes managed (.NET) code
- Language independent
- Applied to compiled code, analyzes Microsoft Intermediate Language
- Applies Microsoft best practices as used in .NET Framework
- Rules also documented in: "Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries"

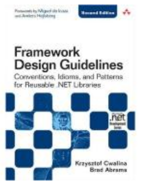

Framework Design Guidelines
Conventions, Idioms, and Patterns for Reusable .NET Libraries
Krzysztof Cwalina
Brad Abrams

MITRE

Improving Development Practices:

Coding Standards – Code Review

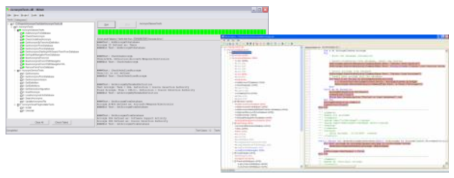
- Preparation requires inspection of code by developer – may uncover defects
- Review by other programmers – leads to sharing of ideas, improved coding techniques
- Review by others may uncover defects or poor techniques
- To be effective, focus should be on determining causes of defects, fixing causes.
- Government audit provides needed assurance on the level of conduct

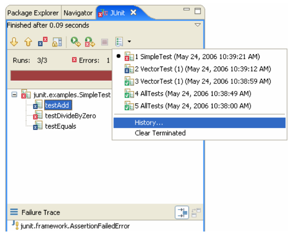

Framework Design Guidelines
Conventions, Idioms, and Patterns for Reusable .NET Libraries
Krzysztof Cwalina
Brad Abrams

MITRE

Improving Development Practices: Automated Unit Testing

- Design Impact
  - Design for Test
  - Test Driven Development
- Tools and Techniques
  - .NET
    - NUnit/NCover/NCover Explorer
    - Visual Studio
  - Java
    - JUnit/Cobertura (etc.)
- Enterprise Impact
  - Extension to Enterprise
  - Uniform Tool Usage
  - Use by Test Organizations



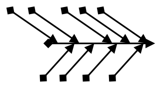

Package Explorer
Navigator
JUnit
Finished after 0.09 seconds
Runs: 3/3
Errors: 1
junit.examples.SimpleTest
testAdd
testDivideByZero
testEquals
1 SimpleTest (May 24, 2006 10:39:21 AM)
2 VectorTest (1) (May 24, 2006 10:39:12 AM)
3 VectorTest (1) (May 24, 2006 10:38:59 AM)
4 AllTests (May 24, 2006 10:38:49 AM)
5 AllTests (May 24, 2006 10:38:00 AM)
History...
Clear Terminated
Failure Trace
junit.framework.AssertionFailedError

MITRE

When the requirement for testing is included, design approaches may be altered to accommodate this. For example, global references hinder testability while passing all data as parameters into a routine and collecting the result as an output makes testing much easier.

Improving Development Practices: Root Cause Analysis

- A CMMI 5 practice area – but this should be a requirement *regardless of CMMI level*.
- Find the cause
  - "Five Whys"
  - Kepner-Trego Problem Analysis
  - IBM: Defect Causal Analysis
- Fix the cause => change the process
- Fix the problem: use the changed process
- Problem: How to Preserve Knowledge?
  - Answer: Classify Root Causes
  - Look for patterns
  - Metrics
    - Statistics
    - Pareto Diagrams



Top level categories :
- 0xxx Planning
- 1xxx Requirements and Features
- 2xxx Functionality as Implemented
- 3xxx Structural Bugs
- 4xxx Data
- 5xxx Implementation
- 6xxx Integration
- 7xxx Real-Time and Operating System
- 8xxx Test Definition or Execution Bugs
- 9xxx Other

Root cause analysis is the key to quality improvement

MITRE

Improving Development Practices:
Root Cause Classification

- Beizer Taxonomy
  - Classification System for Root Causes of Software Defects
  - Developed by Boris Beizer
  - Published in 1990 in “Software Testing Techniques 2nd Edition”
  - Modified by Otto Vinter (around 1998)
  - Based on the Dewey Decimal System
  - Extensible Classification
  - The uniform use of this taxonomy provides an Enterprise view of problem areas in software development.
- Orthogonal Defect Classification
- Defect Causal Analysis

There are other classification methods which might be used:
e.g. Orthogonal Defect Classification – primarily developed by IBM.
However, the Beizer Taxonomy lends itself well to further analysis
(Database queries, Pareto charts, cross-developer comparisons, etc.)

Classifying Root Causes: Beizer* Taxonomy

Top level categories :

- 0xxx Planning
- 1xxx Requirements and Features
- 2xxx Functionality as Implemented
- 3xxx Structural Bugs
- 4xxx Data
- 5xxx Implementation
- 6xxx Integration
- 7xxx Real-Time and Operating System
- 8xxx Test Definition or Execution Bugs
- 9xxx Other

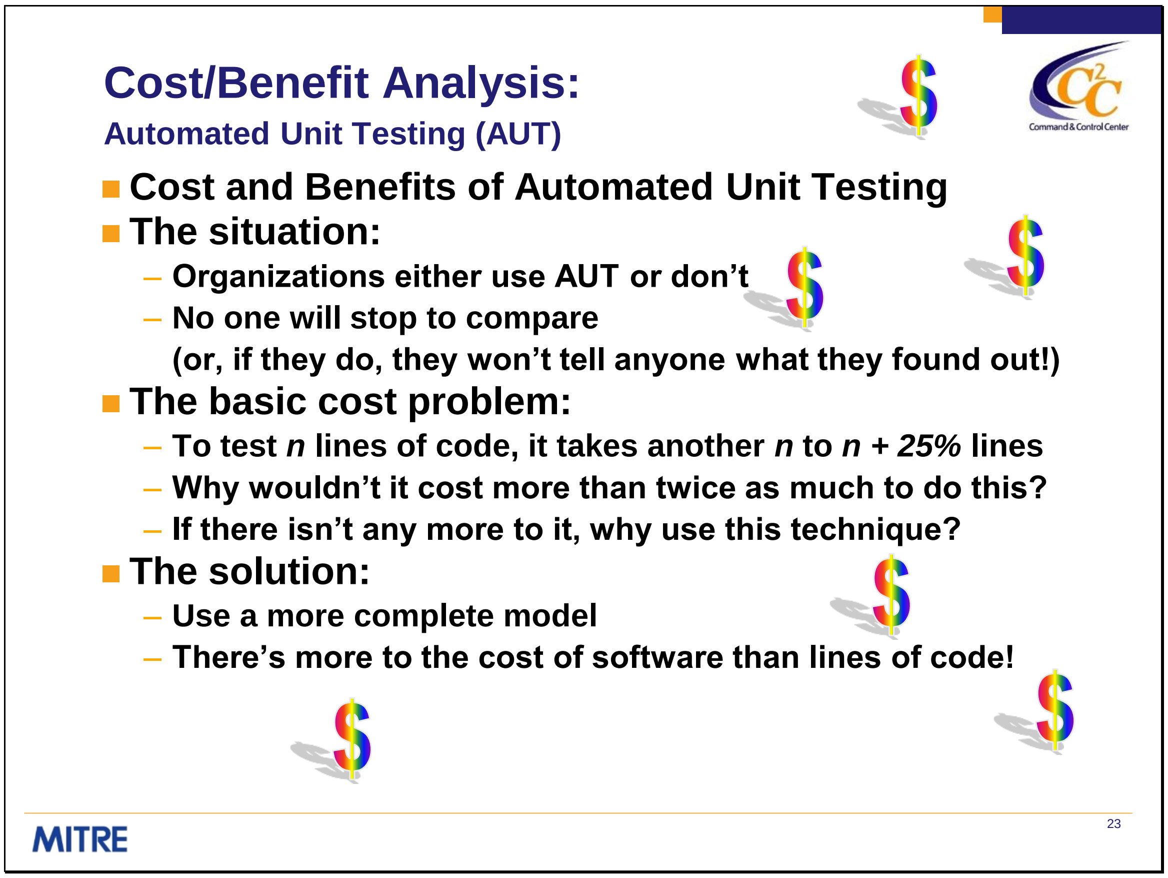
Otto Vinter’s amended version includes more statistics and added detailed categories, with Boris Beizer’s endorsement

* Boris Beizer, "Software Testing Techniques", Second edition, 1990, ISBN-0-442-20672-0

Improving Development Practices:
Software Reuse for .NET

- Extensibility features in .NET
- Microsoft Patterns and Practices
  - Enterprise Library
    - Data Access Application Block
    - Logging Application Block
    - Tracing (Core)
- .NET 3.5 Features
  - Windows Presentation Foundation
  - Windows Communication Foundation
  - Windows Workflow
  - LINQ

How Much Does It Cost?

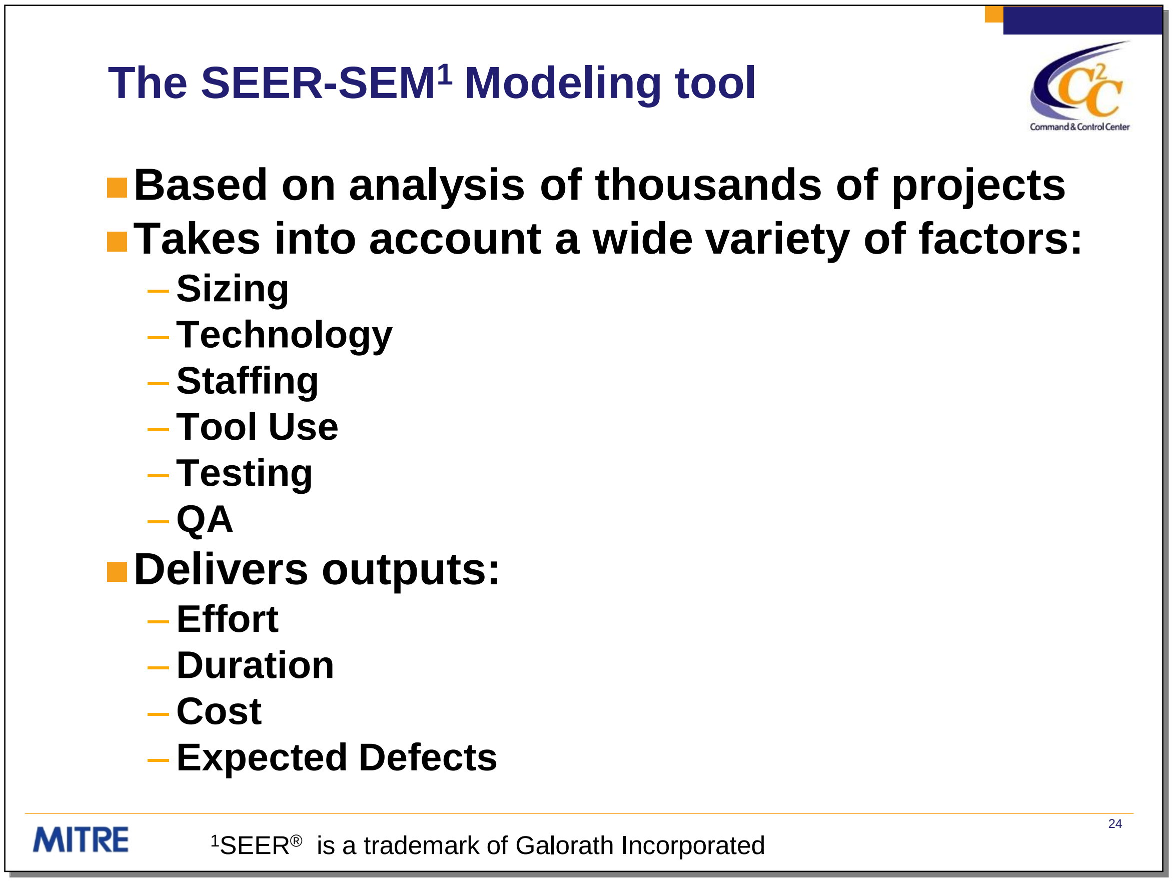


Cost/Benefit Analysis:

Automated Unit Testing (AUT)

- Cost and Benefits of Automated Unit Testing
- The situation:
  - Organizations either use AUT or don't
  - No one will stop to compare
    (or, if they do, they won't tell anyone what they found out!)
- The basic cost problem:
  - To test *n* lines of code, it takes another *n* to *n + 25%* lines
  - Why wouldn't it cost more than twice as much to do this?
  - If there isn't any more to it, why use this technique?
- The solution:
  - Use a more complete model
  - There's more to the cost of software than lines of code!

MITRE



The SEER-SEM[1] Modeling tool

- Based on analysis of thousands of projects
- Takes into account a wide variety of factors:
  - Sizing
  - Technology
  - Staffing
  - Tool Use
  - Testing
  - QA
- Delivers outputs:
  - Effort
  - Duration
  - Cost
  - Expected Defects

MITRE

[1]SEER® is a trademark of Galorath Incorporated

Fischman, Lee. McRitchie, Karen and Galorath, Daniel D. Inside SEER-SEM, CrossTalk Magazine, April 2005

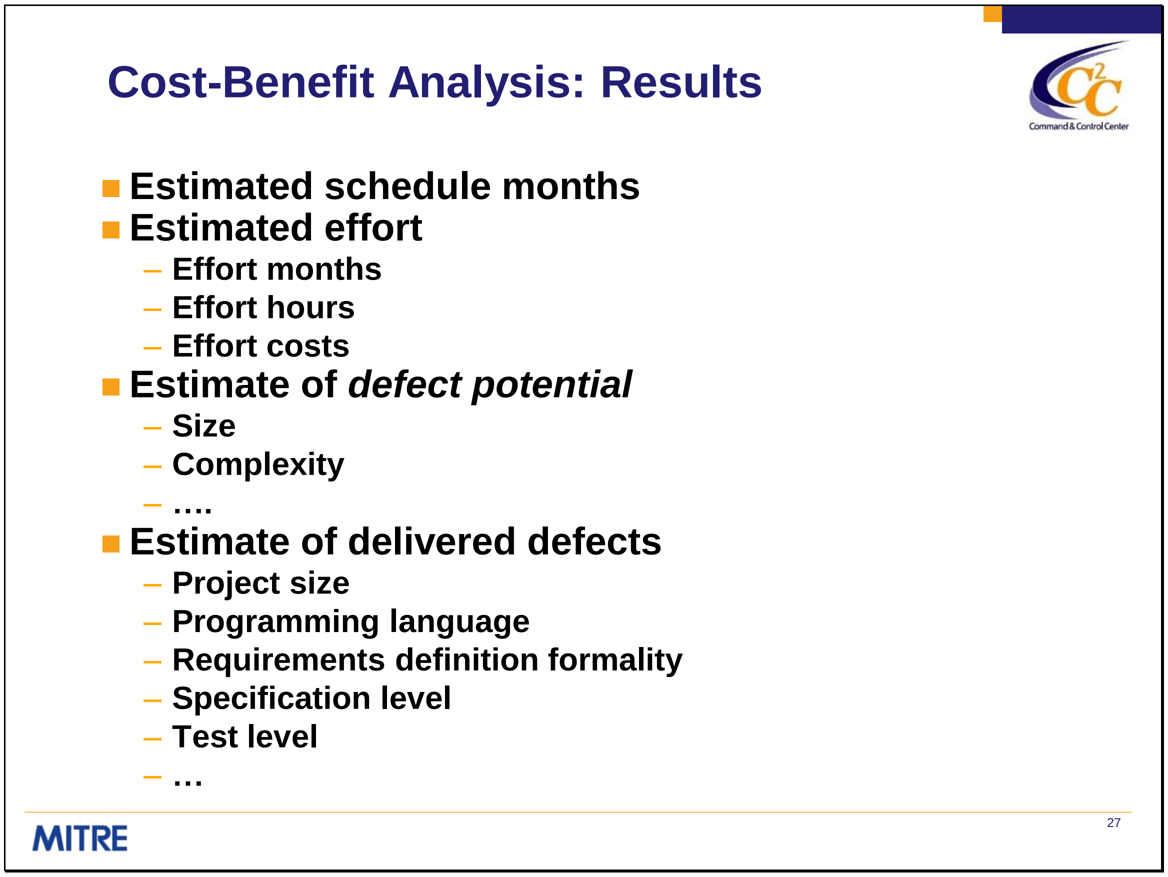
Cost/Benefit Analysis: Technique

- **Consider the cost of defects:**
  - **Legacy defects to fix**
  - **New defects to fix**
  - **Defects not yet fixed (legacy and new)**
- **Model costs using SEER-SEM scenarios**
  - **Cost model reflecting added/modified code**
  - **Comparison among scenarios with varying development techniques**
  - **Schedule, Effort for each scenario**
  - **Probable undetected remaining defects after FQT for each scenario**

The cost of defects is what Crosby calls the “cost of quality”.

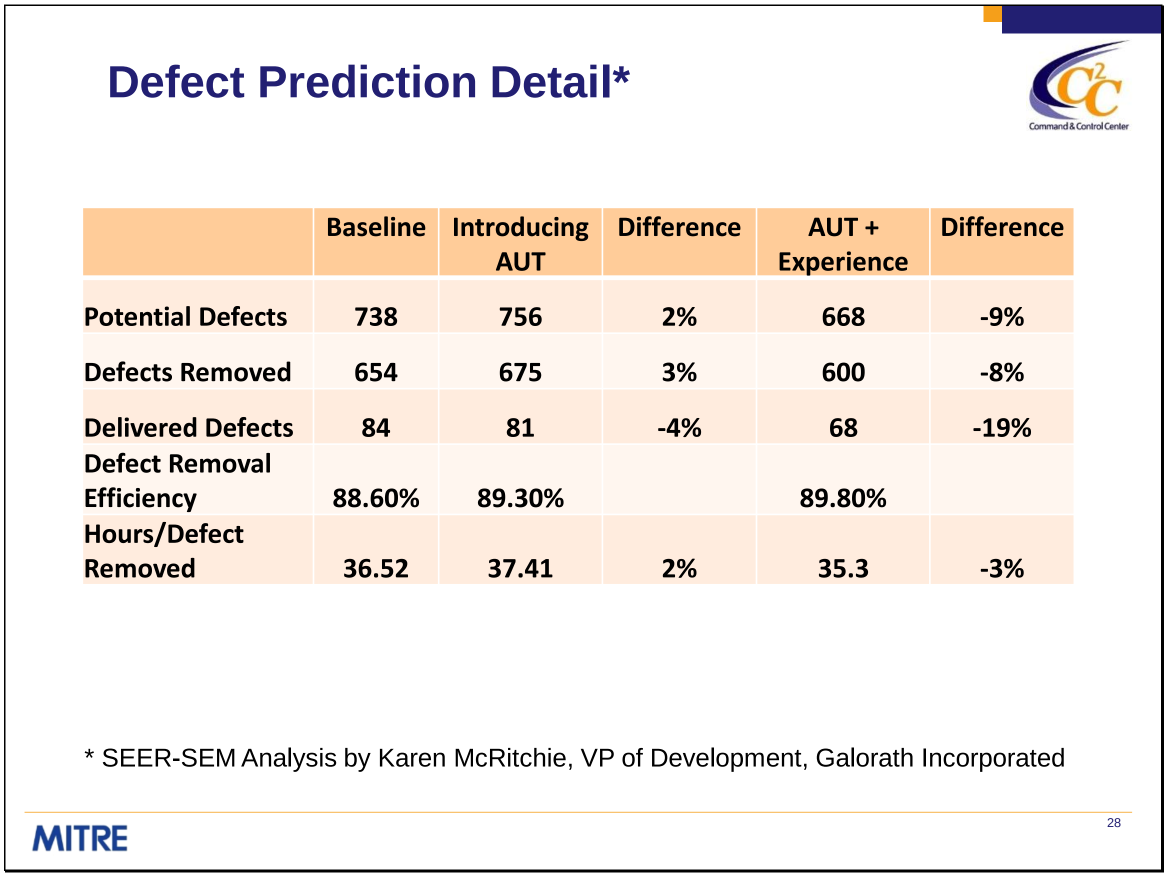
Cost-Benefit Analysis: Example

- **The Project:**
  - **Three major applications**
  - **Two vendor-supplied applications**
  - **Moderate criticality**
- **The cases:**
  - **Baseline: no AUT**
    - **Nominal team experience with environment, tools, practices**
  - **Introducing AUT**
    - **Increases automated tool use parameter**
    - **Decreases development environment experience**
    - **Increases volatility**
  - **Introducing AUT and Added Experience**
    - **Increases automated tool use parameter**
    - **Previous changes to experience and volatility are eliminated**



Cost-Benefit Analysis: Results

- Estimated schedule months
- Estimated effort
  - Effort months
  - Effort hours
  - Effort costs
- Estimate of *defect potential*
  - Size
  - Complexity
  - ….
- Estimate of delivered defects
  - Project size
  - Programming language
  - Requirements definition formality
  - Specification level
  - Test level
  - …



Defect Prediction Detail*

	Baseline	Introducing AUT	Difference	AUT + Experience	Difference
Potential Defects	738	756	2%	668	-9%
Defects Removed	654	675	3%	600	-8%
Delivered Defects	84	81	-4%	68	-19%
Defect Removal Efficiency	88.60%	89.30%		89.80%	
Hours/Defect Removed	36.52	37.41	2%	35.3	-3%

* SEER-SEM Analysis by Karen McRitchie, VP of Development, Galorath Incorporated

When introducing AUT, you see a small increase in the defect removal efficiency. This increase is initially offset by an increase in the overall defect potential that results in an increased number of hours spent removing each defect. However, when you couple AUT with the requisite experience, the increase in defect removal efficiency is boosted by the fact that the overall defect potential is reduced. This reduction in defect potential, combined with the overall effort reduction, quantifies the intuitive adage that the cheapest defect to remove is an avoided defect.

Cost Model*

	Baseline	Introducing AUT	Difference	AUT + Experience	Difference
Schedule Months	17.09	17.41	2%	16.43	-4%
Effort Months	157	166	6%	139	-11%
Hours	23,881	25,250	6%	21,181	-11%
Base Year Cost	$2,733,755	$2,890,449	6%	$2,424,699	-11%
Defect Prediction	84	81	-4%	68	-19%

* SEER-SEM Analysis by Karen McRitchie, VP of Development, Galorath Incorporated

Summary

- The use of known best practices can improve the quality of software
- Better results can be achieved at the same time as lower costs

Questions

MITRE

Selected References

- Spiewak, Rick and McRitchie, Karen. Using Software Quality Methods to Reduce Cost and Prevent Defects, CrossTalk, Dec 2008.
- McConnell, Steve. Code Complete 2. Microsoft Press, 2004.
- Crosby, Philip B. Quality Is Free: The Art of Making Quality Certain. McGraw-Hill Companies, 1979.
- Beizer, Boris. Software Testing Techniques. 2nd ed. International Thomson Computer Press, 1990.

MITRE