CMMI® Institute

APPLYING QUALITY FUNDAMENTALS TO SOFTWARE DEVELOPMENT TO REDUCE COSTS AND PREVENT DEFECTS

NAME: Rick Spiewak

TITLE: Lead Software Systems Engineer

ORGANIZATION: The MITRE Corporation

MITRE

CAPABILITY COUNTS 2017

CMMIINSTITUTE.COM/CONFERENCES

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

BACK TO THE BASICS

- Define quality:
  - "Meeting the requirements."
  - 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

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"

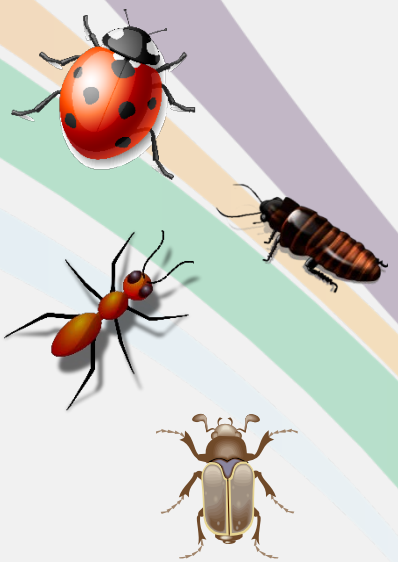
If we want to profit from the work of pioneers in the field of software quality, we owe it to ourselves and them to stand on their shoulders.

PHASES OF SOFTWARE DEVELOPMENT

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

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 stand up

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
- 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 external standards
- Created *ad hoc* by each organization
- No penalty for inadequate standards
- Best practices are often discarded under cost and schedule pressure

SHOULDERS OF GIANTS

- Where do building codes come from?
  - Not from a blank sheet of paper!
- Starting Point: The International Code Council®
  - Vetted by local authorities
  - May rely on standards developed by various independent standards organizations.
- Example- The Building Code of New York State:

  "The Building Code of New York State combines language from the 2006 International Building Code®, and New York modifications developed by the State Fire Prevention and Building Code Council and its Building Code Technical Subcommittee."

A CONCRETE EXAMPLE

- Do we send engineers into a warehouse with a tub of concrete?
- Or – Standing on the shoulders of giants…
  - The American Concrete Institute®
- New York State Building Code:

  "1903.1 General. Materials used to produce concrete, concrete itself and testing thereof shall comply with the applicable standards listed in ACI 318."

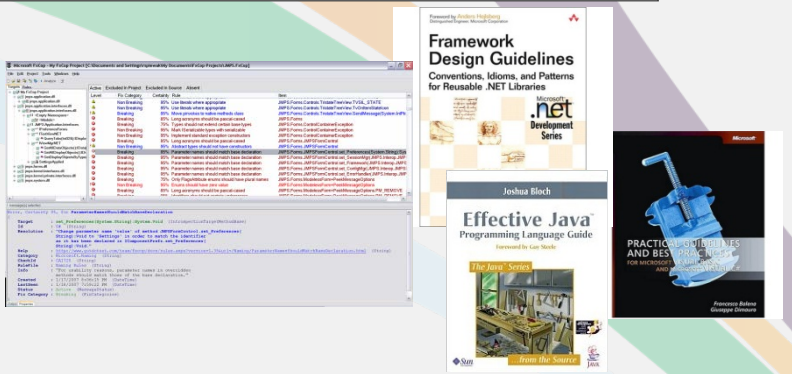
HOW DO WE APPLY THIS TO SOFTWARE?

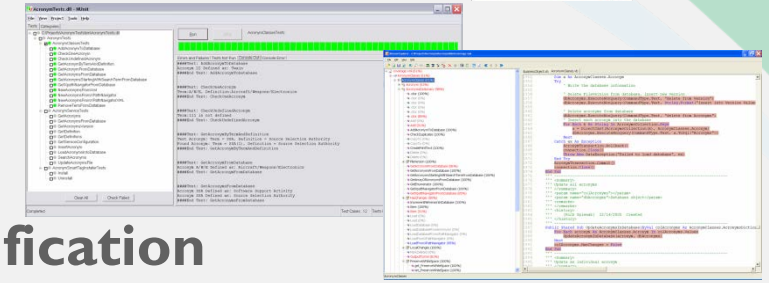
- Don't invent our own standards
- *Identify* industry best practices
- *Enforce* best practices
  - Requirements
  - Rules

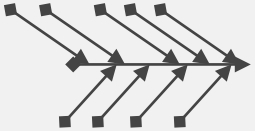
This is how to make sure our software doesn't fall down!

IMPROVING DEVELOPMENT PRACTICES:

- 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

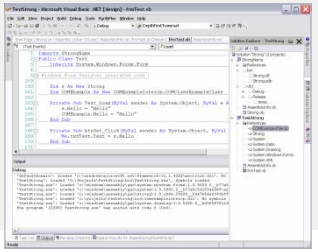






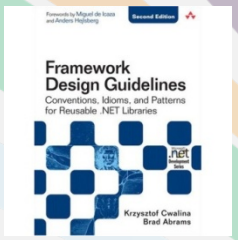
Top level categories :

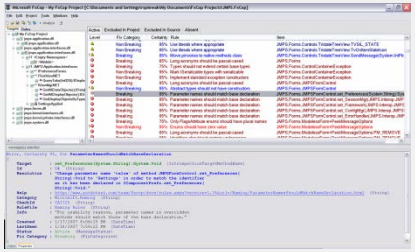
- 0xxx Planning
- 1xxx Requirements and Features
- 2xxx Functionality as Implemented
- 3xxx Structural Bugs
- 4xxx Data
- 5xxx Implementation
- 6xxx Integration
- 7xxx Real-Time and Operating System
- 8xxx Test Definition or Execution Bugs
- 9xxx Other



IMPROVING DEVELOPMENT PRACTICES: UNIFORM CODING STANDARDS

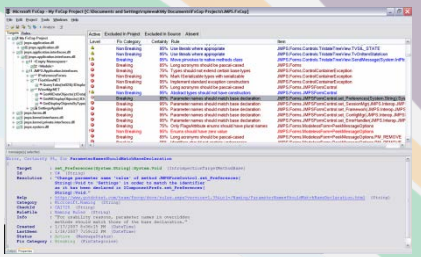
- References
  - .NET
    - Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries
    - Practical Guidelines and Best Practices for Microsoft Visual Basic and C# Developers
  - Java
    - Effective Java Programming Language Guide
    - The Elements of Java Style
-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

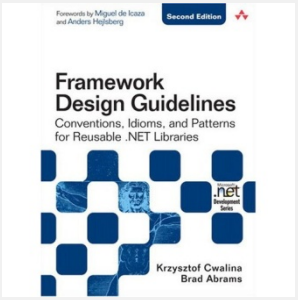




IMPROVING DEVELOPMENT PRACTICES: TOOLS
STATIC ANALYSIS WITH FXCOP FOR .NET

- Microsoft developed free tool
- **Equivalent version in Visual Studio**
- Analyzes managed (.NET) code
- Language independent
- Applied to compiled code
- Applies Microsoft best practices
- Rules also documented in: "Framework Design Guidelines"



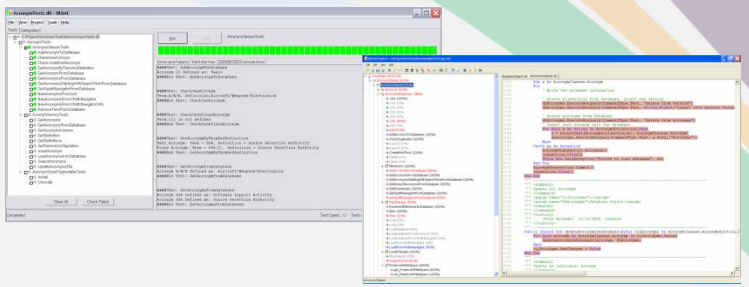


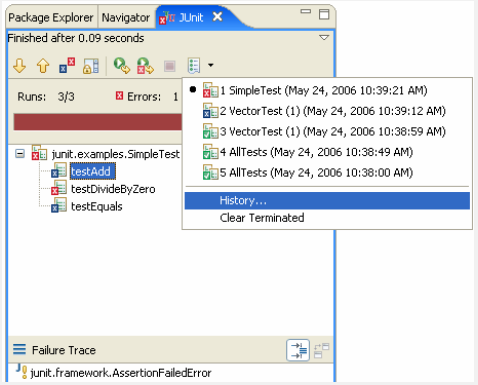
IMPROVING DEVELOPMENT PRACTICES:
CODING STANDARDS – CODE REVIEW

- Preparation
  - inspection of code by developer
  - may uncover defects, inspire changes
- Review by other programmers
  - sharing of ideas, improved techniques
  - uncovers defects or poor techniques
- Determining, fixing causes of defects
- Customer audit
  - provides assurance of execution

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
  - Uniform Developer Usage
  - Use by Test Organizations

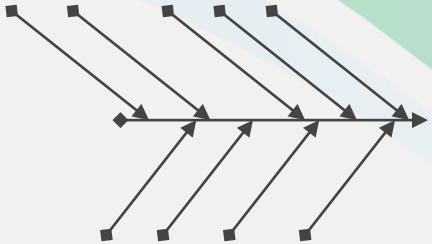




MITRE

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
- How to Preserve Knowledge?
  - 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

IMPROVING DEVELOPMENT PRACTICES: ROOT CAUSE CLASSIFICATION

- **Beizer Taxonomy**
  - Classification of Root Causes of Software Defects
  - Developed by Boris Beizer
  - Published in "Software Testing Techniques 2nd Edition"
  - Modified by Otto Vinter
  - Based on the Dewey Decimal System
  - Extensible Classification
  - "A Beizer categorisation can be performed at a rate of 5 minutes per bug" *
- **Orthogonal Defect Classification**
- **Defect Causal Analysis**

* PRIDE report, section 5.1

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

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 Library Features
  - Windows Presentation Foundation
  - Windows Communication Foundation
  - Windows Workflow
  - Entity Framework
  - LINQ

IMPROVING REQUIREMENTS

- Requirements: a key source of defects
- 1999 – PRIDE study in Denmark
- Key techniques studied:
  - User scenario descriptions
  - Navigational Prototype Usability Testing
- Key results achieved:

  (Comparison between product versions)
  - 27% reduction in error reports
  - 72% reduction in usability issues per new screen
  - Almost 3 times difference in productivity

HOW MUCH DOES IT COST?

COST/BENEFIT ANALYSIS: AUTOMATED UNIT TESTING (AUT)

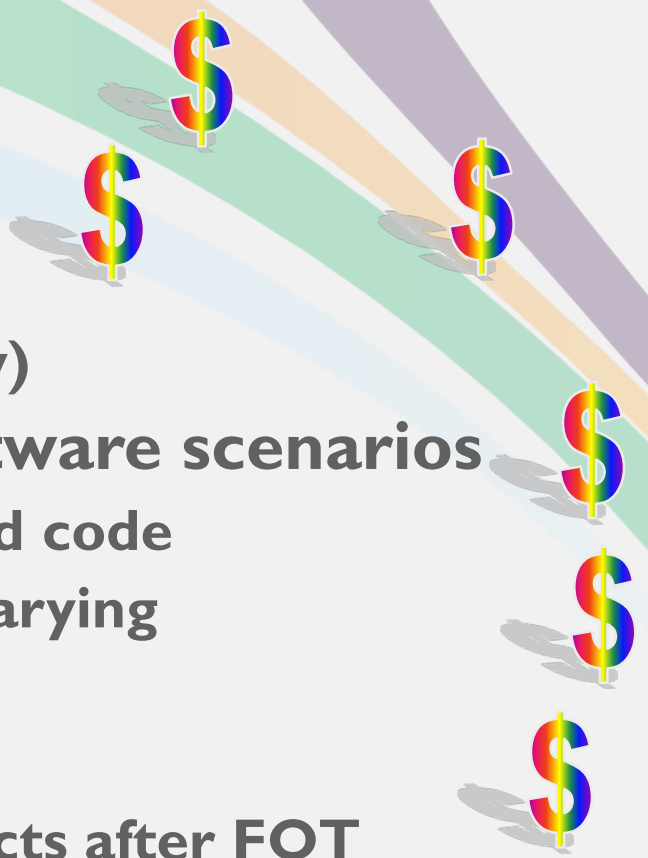
- Cost and Benefits of Automated Unit Testing
- The situation:
  - Organizations either use AUT or don't
  - No one will stop to compare
    (if they do, they won't tell anyone what they found out!)
- The basic cost problem:
  - To test *n* lines of code, it takes *n* to *n* + 25% lines
  - Why wouldn't it cost more 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 cost than lines of code!

THE SEER FOR SOFTWARE[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

[1]SEER® is a trademark of Galorath Incorporated

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 For Software 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 (Formal Qualification Test) per scenario

COST-BENEFIT ANALYSIS: EXAMPLE

- The Project:
  - Three major applications
  - Two vendor-supplied applications
  - Moderate criticality
- The cases:
  - Baseline: no AUT
    - Nominal team experience (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
    - Experience and volatility changes 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 Analysis by Karen McRitchie
VP of Development, Galorath Incorporated

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 Analysis by Karen McRitchie
VP of Development, Galorath Incorporated

DEFECT REMOVAL – CAPERS JONES

- What are the best, proven techniques?
- Optimal Sequence of Software Defect Removal

  From:

  "Software Engineering Best Practices:

  Lessons from Successful Projects in the Top Companies"

  McGraw-Hill, 2010. Chapter 9, pp. 618-619
- Combining these recommended methods "will achieve cumulative defect removal efficiency levels in excess of 95 percent for every software project and can achieve 99 percent for some projects."

Pretest Defect Removal

1. Requirements inspection
2. Architecture inspection
3. Design inspection
4. Code inspection
5. Test case inspection
6. Automated static analysis

Testing Defect Removal

7. Subroutine test
8. Unit test
9. New function test
10. Security test
11. Performance test
12. Usability test
13. System test
14. Acceptance or beta test

WHAT SHOULD WE DO?

- If you develop software:
  - Follow general best practices
  - Add best practices for your technology
  - Formulate your own guidelines but -
    - Stand on the shoulders of giants!!
  - Enforce your guidelines through reviews
- If you contract for software development
  - Examine the practices of competitors
  - Insist on accountability for best practices
  - Trust, but verify!
- If you acquire software for the government
  - This is a whole separate topic! See references.

HOW TO INCORPORATE BEST PRACTICES

- Have/Require a Software Development Plan
  - Processes and practices
  - Tools and Techniques to be used
  - Requirements management
  - Types and frequency of reviews
  - Types of tests
  - Configuration and release management
  - Well-Defined Deliverables
- Have/Require a Software Test Plan
  - Development Test
  - QA Testing
  - Acceptance Test

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
- If you want these benefits, you have to require best practices!

QUESTIONS

SELECTED REFERENCES

- Spiewak, Rick and McRitchie, Karen. Using Software Quality Methods to Reduce Cost and Prevent Defects, CrossTalk, Dec 2008.

  http://www.crosstalkonline.org/storage/issue-archives/2008/200812/200812-Spiewak.pdf
- 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.
- Jones, Capers. Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies. McGraw-Hill Companies, 2010.
- Vinter O., S. Lauesen & J. Pries-Heje. A Methodology for Preventing Requirements Issues from Becoming Defects (1999) http://www.ottovinter.dk/Finalrp3.doc
- Spiewak, R. et al. Applying the fundamentals of quality to software acquisition. 2012 IEEE SysCon http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6189447