2017-03

Be Careful What You Pay For: Applying the Fundamentals of Quality to Software Acquisition

Spiewak, Rick

Monterey, California. Naval Postgraduate School

Be Careful What You Pay For:
Applying the Fundamentals of Quality to Software Acquisition

14th Annual Acquisition Research Symposium

10 – 11 May 2017

Rick Spiewak
The MITRE Corporation
rspiewak@mitre.org

MITRE

How to acquire quality software

- Define what we mean by quality
- Understand how quality is achieved
- Apply this to the acquisition process:
  - Specify quality processes
  - Select the right developer
  - Follow the acquisition process rules

What is Software Quality?:

An Application of the Basic Principles of Quality Management

Spoiler alert:

I can’t do this without showing you some of my favorite quotes first

What is Software Quality?:

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?:

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?:

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."
  - **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

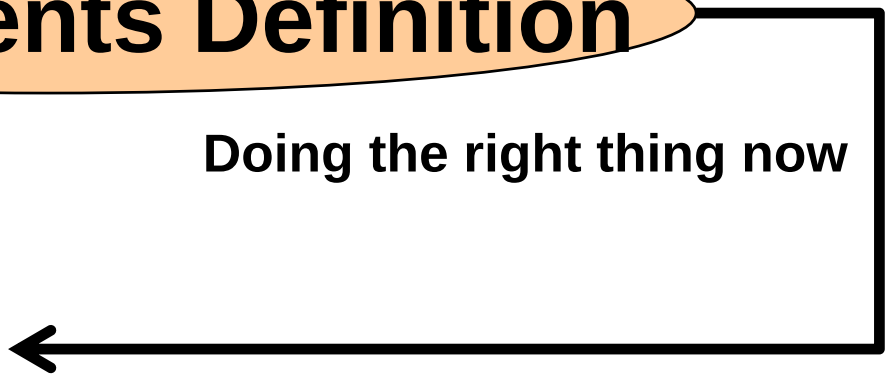
Ways to Get Started

- **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

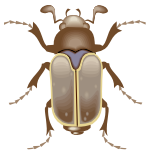
- **Requirements Definition**
- Architecture
- Design
- Construction
- Testing
- Documentation
- Training
- Deployment
- Sustainment



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 is a separate activity**
- **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

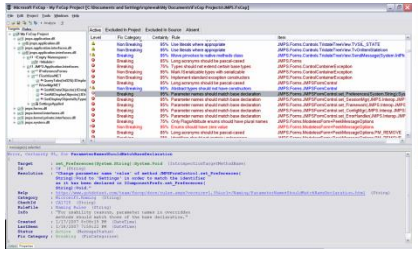
- There is a lack of uniformity and standards
- Historically, these are created *ad hoc* by each organization
- There is no penalty for inadequate standards
- Best practices are often discarded under cost and schedule pressure

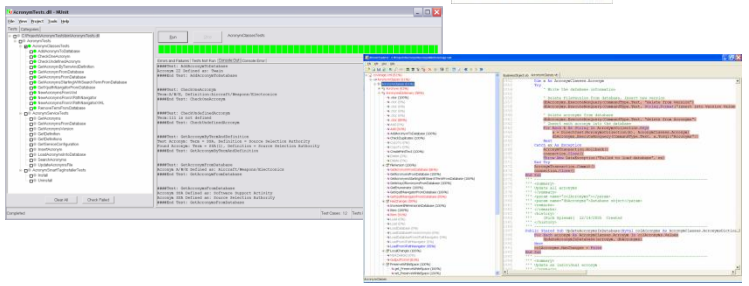
How Do We Fix This?

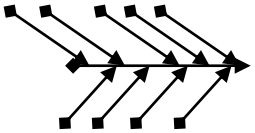
- We must *identify* and *implement* industry best practices
- We must *enforce* best practices
  - Requirements (acquisition)
  - Rules (implementation)
- This is the way to make sure our software doesn't burn up or fall down!

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



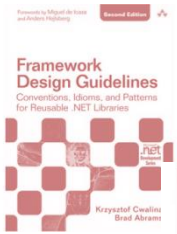


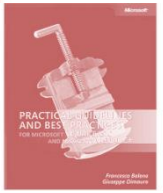


Top level categories :

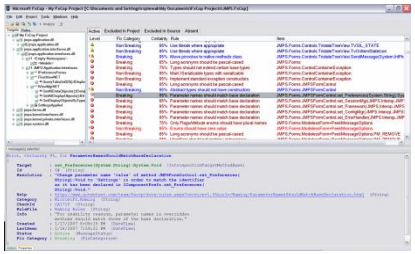
- 0xxx Planning
- 1xxx Requirements and Features
- 2xxx Functionality as Implemented
- 3xxx Structural Bugs
- 4xxx Data
- 5xxx Implementation
- 6xxx Integration
- 7xxx Real-Time and Operating System
- 8xxx Test Definition or Execution Bugs
- 9xxx Other

Improving Development Practices:
Uniform Coding Standards



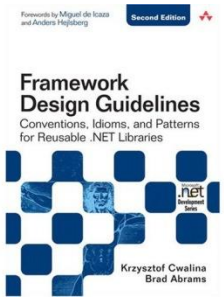


- References
  - .NET
    - Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries
    - Practical Guidelines and Best Practices for Microsoft Visual Basic and C# Developers
  - Java
    - Effective Java Programming Language Guide
    - The Elements of Java Style
-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



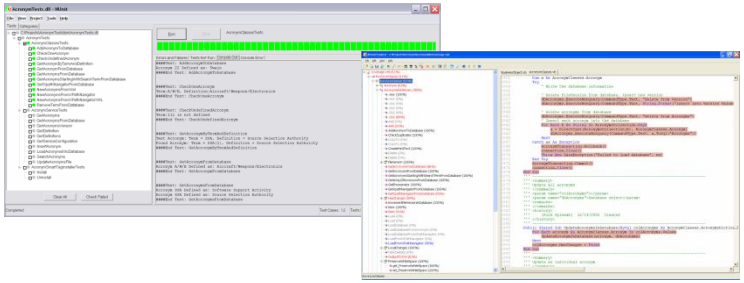
Improving Development Practices:
Coding Standards – Code Review

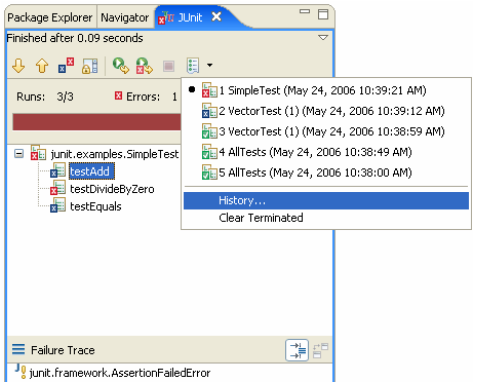
- Preparation requires inspection of code by developer – may uncover defects
- Review by other programmers – leads to sharing of ideas, improved coding techniques
- Review by others may uncover defects or poor techniques
- To be effective, focus should be on determining causes of defects, fixing causes.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



Improving Development Practices:
Automated Unit Testing

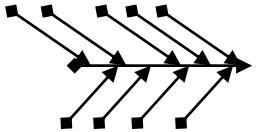
- Design Impact
  - Design for Test
  - Test Driven Development
- Tools and Techniques
  - .NET
    - NUnit/NCover/NCover Explorer
    - Visual Studio
  - Java
    - JUnit/Cobertura (etc.)
- Enterprise Impact
  - Extension to Enterprise
  - Uniform Tool Usage
  - Use by Test Organizations





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

Acquisition: How to Require Best Practices

- Two areas of focus:
  - Instructions for Proposal Preparation
    - IFPP, or Section L
  - Evaluation Criteria
    - EC, or Section M
- Why:
  - You can't use what you haven't asked for!
  - It's too late to adjust your evaluation criteria after the proposals are delivered.
- "Standing on the shoulders…"
  - There is existing work you can leverage to build your RFP
  - There is precedent for using these criteria

Shoulders of Giants

- **USAF Weapon Systems Software Management Guide, August 2008.**
  - **Appendix C Example Software Content for RFP Section L**
  - **Appendix D Example Software Content for RFP Section M**
- **Guidebook for Acquisition of Naval Software Intensive Systems, September 2008.**
  - **7.5 Section L - Instructions, Conditions, and Notices to Offerors**
  - **7.6 Section M - Evaluation Factors for Award**

There is a lot of useful information in these guides – here we're focusing on just these two areas.

Key Recommendations: Instructions for Proposal Preparation

- Require a Software Development Plan (SDP)
  - Describes the offeror's approach to software development
  - Tools and techniques to be used:
    - Development
    - Unit testing
    - Component testing
    - Integration
    - Configuration management
    - Managing defect reports and analysis
    - Root cause analysis
- Require that this be used as the basis for a final plan
  - SDP to be made available as a deliverable item, subject to review and approval
  - Provisions of the plan to be followed as described

Key Recommendations: Evaluation Criteria

- Number and type of peer reviews
- Use of automated unit testing, code coverage
- Use of automated syntax analysis tools
- Comprehensiveness of integration and test methods
- Readiness requirements for code check-in
  - Unit test
  - Syntax analysis
  - Peer Reviews
- Configuration Management and Source Code Control
- Use of Root Cause Analysis
- Code Re-use
  - Sources
  - Quality assurance
  - Government rights

How to Rate the SDP

Parameter/rating	Unacceptable	Marginal	Acceptable	Superior
The number and type of peer reviews	none	1 (any)	2 (design,code)	3 or more (requirements, design,code,test)
The use of automated unit testing including test coverage requirements	none	unit tests written after manual testing or only on selected code	automated tests 75% code coverage on new or modified code	automated tests 85% or more code coverage on all delivered code. The use of Test Driven Development.
The use of automated syntax analysis tools and adherence to the rules incorporated by them	none	used selectively or with heavily modified rules	used consistently with standard rules	additional rules or tools specific to security analysis

How to Rate the SDP – Cont'd

Parameter/rating	Unacceptable	Marginal	Acceptable	Superior
The comprehensive-ness of integration and test methods including continuous integration tools if used	ad-hoc	formal integration and test	automated processes applied periodically	continuous integration including syntax analysis and unit tests
The use of readiness requirements such as unit test and syntax analysis for code check-in	none	individual manual testing	integrated testing by developer	Automated part of check-in and continuous integration process

How to Rate the SDP – Cont'd

Parameter/rating	Unacceptable	Marginal	Acceptable	Superior
Configuration management and source code control tools and techniques	manual/ paper trail	by individual developer	system-wide repository	managed tool with pre-check-in requirements
The extent to which root cause analysis of defects is part of the development process	none	"red-team" only	serious defects	routine periodic analysis of defect pool
The selection of software source code to be reused, replaced, or re-written from previous implementations	none or no response	replacement with contractor's previous work	rework of selected items showing good knowledge of base software	innovative approach to maximum reuse and modernization

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
- By including an evaluation of development practices at the proposal evaluation stage, these can be used as source selection criteria

Selecting the right developer is the starting point for getting the right results

Questions

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.
- Jones, Capers. Software Engineering Best Practices. McGraw-Hill, 2010
- USAF Weapon Systems Software Management Guide, August 2008.
- Guidebook for Acquisition of Naval Software Intensive Systems, September 2008.

Backup

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**

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

How Much Does It Cost?

Cost/Benefit Analysis Example:

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?

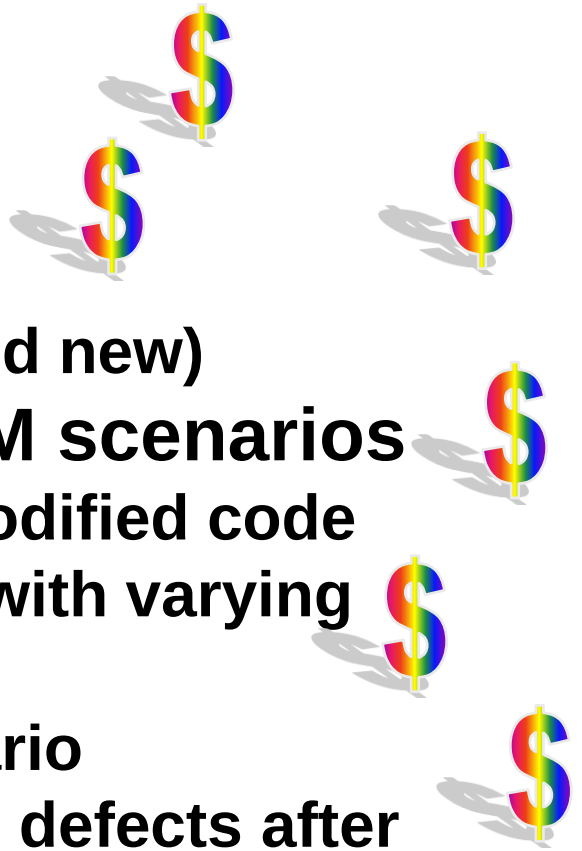
- The solution:
  - Use a more complete model
  - There's more to the cost of software than lines of code!

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

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



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

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