Quality

or

(Is my Software Any “Good”?)

QAI Mid-Atlantic Software Quality and Program Management Conference 2011

Lawrence I. Baker (Professor of Engineering Management)
Top Software Issues

- The impact of requirements upon software is not consistently quantified and managed in development or sustainment.
- Fundamental system engineering decisions are made without the full participation of software engineering.
- Software life-cycle planning and management by acquirers, developers, and suppliers is often ineffective.
- The quantity and quality of software engineering expertise is insufficient to meet the demands of the market, including the government and the defense industry.
- Traditional software verification techniques are costly and ineffective for dealing with the scale and complexity of modern systems.
- There is a failure to assure correct, predictable, safe, secure execution of complex software in distributed environments.
- Inadequate attention is given to total lifecycle issues for COTS/NDI impacts on lifecycle cost and risk.
Quality

What is it anyway?
Does it just happen?
Can it be measured?
Do we just know it when we see it?
Overview

• What is Software Quality?
• How do I achieve Software Quality?
• Software Quality Initiatives
  – Methodology
  – Capabilities/Requirements Definition
  – Measurement
  – Process Maturity
• System/Software Testing
• Best Practices
Definitions for Software Quality

- The degree to which a system, component, or process meets specified requirements
- The degree to which a system, component, or process meets customer or user needs or expectations (no undesirable properties)

Other Definitions

- Lack of bugs
- Adhering to software quality factors ("ilities")

Questions

- What makes defining Software Quality difficult?
- When do software quality problems show up?
Software Quality - Distinctions

• **Software Quality Program:**
  – the approach (both technical and management) to influence and determine the level of quality achieved in a software product

• **Software Quality Evaluation**
  – a set of assessments and measurements that is done throughout the life cycle to evaluate products, documentation and processes

• **Software Quality Assurance (SQA)**
  – Generally a functional entity within an organization charged with some aspect of the overall SQE effort. A wide variety of organizational schemes can be used to staff the SQA effort
THE ESSENCE:
Does it do what it is supposed to?

“Software problems and defects are among the few direct measurements of software processes and products. Problem and defect measurements also are the basis for quantifying several significant software quality attributes, factors, and criteria—reliability, correctness, completeness, efficiency, and usability among them.”

Software Quality Measurement, SEI-92-TR-22
Software Quality Factors

- **Correctness**
  - Does the software do what I want?
- **Portability**
  - Can I use it on other machines?
- **Efficiency**
  - Will it run on my hardware properly?
- **Reliability**
  - Does the software accurately do what I want all of the time?
- **Expandability**
  - Can I add new functionalities?
- **Reusability**
  - Will I be able to reuse some or all of it?
- **Flexibility**
  - Can I change it?
- **Survivability**
  - If some of the system breaks will the software continue to function?
- **Integrity**
  - Is the software secure?
- **Verifiability**
  - Can I test it?
- **Interoperability**
  - Can it interface with other systems?
- **Usability**
  - Can I run the software?
- **Maintainability**
  - Can I fix it?
- **Validation**
  - Can I validate what it does?
Quality Program

- A quality program includes a management process that is capable of ensuring the following key activities:
  - Establish capable processes
  - Continuously improve processes
  - Monitor and control critical processes and product variation
  - Establish mechanisms for field product performance feedback
  - Implement an effective root-cause analysis and corrective action system

- Program Managers should allow contractors to define and use their preferred quality management processes that meet required program support capabilities.

- Program Managers should not (typically) require third party certification or registration of a supplier’s quality system

If you don’t plan for Quality ‘up front and early’ you certainly won’t find it at the end.
Software Quality Initiatives

• **Systems/Software Engineering**
  - The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. [IEEE Std 610.12-1990]

• **Lean Six Sigma**
  - A disciplined, data-driven approach and methodology for eliminating defects in any process -- from manufacturing to transactional and from product to service.

• **Continuous Process Improvement**
  - A strategic approach for developing a culture of continuous improvement in the areas of reliability, process cycle times, costs in terms of less total resource consumption, quality, and productivity.

• **Software Clean Room Engineering**
  - Combines formal methods of requirements and design with statistical usage testing to produce software with nearly none or no defects.
Software Quality Initiatives
(continued)

• **Software Assurance**
  – The objective of software assurance is to establish a basis for gaining justifiable confidence (trust, if you will) that software will consistently demonstrate one or more desirable properties. These include such properties as quality, reliability, correctness, dependability, usability, interoperability, safety, fault tolerance, and security.

• **Architecture**
  – The organizational structure of a system or component [IEEE Std 610.12-1990]
  – The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. [IEEE Std 1471-2000]

* Information Assurance Technology Analysis Center (IATAC)
One Practical Definition of “Software Quality”

- Low Defect Potentials (< 1 per function point)
- High Defect Removal (> 95%)
- Unambiguous, Stable Requirements (< 2.5% change)
- Explicit Requirements Achieved (> 97.5% achieved)
- High User Satisfaction Ratings (> 90% “Excellent”)
  - Installation
  - Ease of Learning
  - Ease of Use
  - Functionality
  - Compatibility
  - Error Handling
  - User Information (screens, manuals, etc)
  - Customer Support
  - Defect Repairs

Software developers and acquirers at firms that the GAO investigated used three fundamental management strategies to ensure the delivery of high-quality products on time and within budget: working in an evolutionary environment, following disciplined development processes, and collecting and analyzing meaningful metrics to measure progress.
ISO 9000

• ISO 9000, Quality Management and Quality Assurance Standards
  • Provides guidelines as to which document to use and how to use it.
  • Use of ISO 9001, 9002 or 9003 depends on business structure

• ISO 9001, Quality Systems
  • Model for Quality Assurance in Design/Development, Production Installation, and Servicing

• ISO 9002, Quality Systems
  • Model for Quality Assurance in Production and Installation

• ISO 9003, Quality Systems
  • Model for Quality Assurance in Final Inspection and Test

• ISO 9004, Quality Management and Quality Systems Element
  • Along with ISO 9000, ISO 9004 is an advisory document.

ISO 9001 gives the requirements for what the organization must do to manage processes affecting quality of its products and services.
ISO 9001 - Quality Management Requirements

4.0 General Requirements
   4.1 Develop your Quality Management System (QMS)
   4.2 Document your QMS

5.0 Management Requirements
   5.1 Management Commitment
   5.2 Customer Focus
   5.3 Quality Policy
   5.4 QMS Planning
   5.5 Responsibility, Authority and Communication
   5.6 Management Review

6.0 Resource Requirements
   (Includes Human Resources, Infrastructure and Work Environment)

7.0 Realization Requirements
   (Includes customer communications, purchasing and product identification)

8.0 Remedial Requirements
   8.1 Establish Monitoring and Measurement Processes
   8.2 Carry out monitoring and measurement (customer satisfaction)
   8.3 Control of Nonconforming Product
   8.4 Collect and Analyze Quality Management Data
   8.5 Make Improvements and Take Remedial Action

Software Quality Metrics

1) Establish Requirements
   a) Identify list of possible quality requirements
   b) Determine list of quality requirements
   c) Assign a direct metric to each quality requirement

2) Identify Metrics
   a) Apply the software quality metrics framework
   b) Perform a cost-benefit analysis
   c) Gain commitment to the metrics

3) Implement Metrics
   a) Define the data collection procedures
   b) Prototype the measurement process
   c) Collect the data and compute the metric values

4) Analyze Results
   a) Interpret the results
   b) Identify software quality
   c) Make software quality predictions
   d) Ensure compliance with requirements

5) Validate Metrics
   a) Apply the validation methodology
   b) Apply the validity criteria
   c) Apply the validation procedures
   d) Document results

IEEE Standard for a Software Quality Metrics Methodology
IEEE Std 1061-1998
Software Quality and Capability

- **Practical Software and Systems Measurement (PSM)**
  - Best practices within the software/system acquisition and engineering communities.
  - Goal is to provide managers with the information needed to meet cost, schedule, and technical objectives on programs.

- **Control Objectives for Information and related Technology (COBIT)**
  - Provides good practices across a domain and process framework.
  - Practices designed to help optimize IT-enabled investments, ensure service delivery and provide a measure against which to judge when things do go wrong.

- **Information Technology Infrastructure Library (ITIL)**
  - Provides international best practices for IT service management.
  - Consists of a series of books giving guidance on the provision of quality IT services, and on the accommodation and environmental facilities needed to support IT.

- **SPICE (SW Process Improvement and Capability Determination) (ISO/IEC 15504)**
  - An international standard for software process assessment.
  - Derived from process lifecycle standard ISO 12207 and ideas of maturity models like Bootstrap, Trillium and the CMM.
<table>
<thead>
<tr>
<th>Immature Process</th>
<th>Mature Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad hoc; process improvised by practitioners and their management; Not rigorously followed or enforced</td>
<td>Consistent with the way work actually gets done</td>
</tr>
<tr>
<td>Highly dependent on current practitioners</td>
<td>Defined, documented, and continuously improving</td>
</tr>
<tr>
<td>Low visibility into progress and quality</td>
<td>Supported visibly by management and others</td>
</tr>
<tr>
<td>Product functionality and quality may be compromised in order to meet schedule</td>
<td>Well controlled - process fidelity is audited and enforced</td>
</tr>
<tr>
<td>Use of new technology risky</td>
<td>Product and process measurement used</td>
</tr>
<tr>
<td>Excessive maintenance costs</td>
<td>Disciplined use of technology</td>
</tr>
<tr>
<td>Quality difficult to predict</td>
<td></td>
</tr>
</tbody>
</table>
Capability Maturity Model – Integration (CMMI)

- **Capability Maturity Model® Integration (CMMI)**
  - A collaborative effort sponsored by the Office of the Secretary of Defense/Acquisition and Technology (OSD/A&T) Systems Engineering with participation by government, industry, and the Software Engineering Institute (SEI)
  - Objective is to develop a product suite that provides industry and government with a set of products to support process and product improvement
  - It can be used to guide process improvement across a project, a division, or an entire organization

- **Benefits of implementing process improvement:**
  - The quality of a system is highly influenced by the quality of the process used to acquire, develop, and maintain it.
  - Process improvement increases product and service quality as organizations apply it to achieve their business objectives.
  - Process improvement objectives are aligned with business objectives.
  - CMMI maturity level can often be a good predictor of whether a software development project will incur cost and schedule overruns

Software Engineering Institute, Carnegie Melon  http://www.sei.cmu.edu/cmmi
CMMI Overview

CMMI Staged Maturity Levels

- Level 1 Initial: Process unpredictable, poorly controlled and REACTIVE
- Level 2 Managed: Process characterized for PROJECTS and is MANAGED
- Level 3 Defined: Process characterized for the ORGANIZATION and is PROACTIVE
- Level 4 Quantitatively Managed: Level 4 Quantitatively Managed
- Level 5 Optimizing: Focus on CONTINUOUS process improvement

Process QUANTITATIVELY measured and controlled
## Typical Costs of Software Fixes

<table>
<thead>
<tr>
<th>Lifecycle Software Development Activity</th>
<th>Initial $ Spent</th>
<th>Errors Introduced</th>
<th>Errors Found</th>
<th>Relative Cost of Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Analysis</td>
<td>5%</td>
<td>55%</td>
<td>18%</td>
<td>1.0</td>
</tr>
<tr>
<td>Design Analysis</td>
<td>25%</td>
<td>30%</td>
<td>10%</td>
<td>1.0 – 1.5</td>
</tr>
<tr>
<td>Testing Activities</td>
<td>60%</td>
<td>10%</td>
<td>50%</td>
<td>1.5 – 5.0</td>
</tr>
<tr>
<td>Documentation</td>
<td>10%</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Support</td>
<td>---*</td>
<td>5%</td>
<td>22%</td>
<td>10-100</td>
</tr>
</tbody>
</table>

* After a system is fielded, support costs are typically 50-70% of total system life-cycle costs.
Requirements Development

(continued)

- Requirements Validation
- Functional Verification
- Design Verification

#1: Define Customer Expectations

#2: Define Project & Enterprise Constraints

#3: Define External Constraints

#4: Define Operational Scenarios

#5: Define Measures of Effectiveness

#6: Define System Boundaries

#7: Define Interfaces

#8: Define Utilization Environments

#9: Define Life-Cycle Process Concepts

#10: Define Functional Requirements

#11: Define Performance Requirements

#12: Define Modes of Operation

#13: Define Technical Performance Measures

#14: Define Design Characteristics

#15: Define Human Systems Integration

Establish Requirements Baseline

Operational View

Functional View

Design View

Verification & Validation

Adapted from IEEE 1220
Software Quality Assurance Plan

- Purpose and Scope (SIs and life cycle covered)
- Management (tasks and responsibilities)
- Documentation (types and evaluation criteria)
- Standards, practices, conventions and metrics
- Reviews and Audits (which + how conducted)
- Testing Activities (not described elsewhere)
- Problem Reporting and Corrective Action
- Tools, techniques and methodologies
- Code Control (part of SW Configuration Management Plan)
- Media Control (part of SW Configuration Management Plan)
- Supplier Control (SQAP flow-down)
- Record collection, maintenance, retention
- Training (needed to meet SQAP needs)
- Risk Management (methods and procedures)

* format based on IEEE Standard 730 and 730.1

Links to other plans
Test & Evaluation Master Plan (TEMP)
Documents the overall structure and objectives of the T&E program
Includes critical operational effectiveness and suitability parameters for [among other items]...software and computer resources...it addresses software maturity and the degree to which software design has stabilized.

Software Development Plan (SDP)
- Coding & Unit Testing
- Unit Integration & Test
- SW Qualification Testing
- SW/HW Integration & Test
- System Qualification Testing
- Software Quality Assurance
- Corrective Action Process

Software Test Plan (STP)
- IDs Test Items
- ID’s Personnel Resources
- ID’s Test Environment
- Provides Test Schedules
- Traces to Requirements

DT: system technical performance substantiated?
OT: satisfy ‘real-world’ needs?
Model-Based Testing

- Build the model
- Generate expected inputs
- Generate expected outputs
- Run tests
- Compare actual outputs with expected outputs
- Decide on further actions

From: DACS Gold Practice Document Series GP-34 V 1.1 Model-Based Testing
Software Testing Problem Reporting

**Priority Classification Scheme**

**Priority 1:** Prevents Mission Accomplishment or jeopardizes safety or other “critical” requirement

**Priority 2:** Adversely affect Mission Accomplishment or cost, schedule, performance or software support + no work-around exists

**Priority 3:** Adversely affect Mission Accomplishment or cost, schedule, performance or software support + a work-around exists

**Priority 4:** User/Operator or support inconvenience

**Priority 5:** All other problems

**Classification Scheme (J-STD-016)**

“Alternate category and priority schemes may be used by developers.”

- Project Plans
- Operational Concept
- System/Software Requirements
- Design
- Code
- Databases/data files
- Test Plans, Descriptions, Reports
- User, Operator or Support Manuals
- Other Software Products
Human-Based Software ‘Testing’

**Desk Checking**
- Independent programmer(s) work through a hard copy of the source code

**Pair Programming**
- Writing the source code of a program in teams of two

**Team Review**
- Moderator leads team through line-by-line review

**Walkthrough**
- Step-by-step review of a specification, usability features or design before it is handed off to the technical team for development

**Formal Inspection**
- In-process technical reviews of a product of the software life cycle to find and eliminate defects

- Process steps include:
  - (1) Planning,
  - (2) Overview,
  - (3) Preparation,
  - (4) Inspection meeting,
  - (5) Rework, and
  - (6) Follow-up

**Increasing (as you go up)**
- Documentation, Level of Effort, Defect Removal, Overall Effectiveness

- 30-40% Defect Removal
- 70% + Defect Removal

30-40% Defect Removal: 70% + Defect Removal
Computer-Based Software Testing

- **Acceptance Testing**
  - Verifies that product (usually externally developed) meets customer specified requirements

- **Black Box Testing**
  - No knowledge of the internal workings of the tested item, usually functional

- **Functional Testing**
  - Validates that an application conforms to its specifications and correctly performs all its required functions (feature by feature validation)

- **Integration Testing**
  - Modules are combined and tested as a group

- **Regression Testing**
  - Allows a consistent, repeatable validation of each new release of a product

- **Smoke Testing**
  - Tests only the major functions of a piece of software without bothering with finer details

- **Unit Testing**
  - Test to ensure correct behavior of components prior to system integration

- **White Box Testing**
  - Based on an analysis of internal workings/structure of software
Software Development and Testing Overview

System Requirements Analysis
System Design
Software Requirements Analysis
Software Design
Software Unit Coding
Software Unit Testing

SI
Software Units

SI
Software Units

HI
Software Units

SI
Software Units

DT and OT&E Testing
System Qualification Testing
Subsystem Integration & Testing
Hardware/Software Integration and Test
Software Item (SI) Qualification Testing
Software Unit Integration and Testing
## Test Readiness

### Inputs/Entry Criteria
- Requirements identified.
- Traceability of tests to the requirements has been established.
- All software test procedures have been completed.
- All testing objectives identified.
- Applicable documentation, procedures and plans are complete and under control.
- Method of documenting and dis-positioning “test anomalies” is acceptable.

### Outputs/Exit Criteria
- Software Tests are defined, verified, and baselined.
- Testing is consistent with any defined incremental approaches.
- Test facilities available and appropriate.
- Software under configuration control.
- Lower-level software tests done.
- Software metrics support readiness to start testing.
- Software problem report system is defined and implemented.
- Software test baseline established.
- Development estimates updated.
- Non-tested requirements are identified.
Verification and Validation (V&V)

**Quality** - The degree to which a set of inherent characteristics fulfills *requirements*.

**Verification**
Ensuring that the system is well engineered.

Answers Question:
“Did I build the system right?”

**Validation**
Ensuring that the software meets the users’ needs.

Answers Question:
“Did I build the right system?”
Independent Verification & Validation (IV&V)

- IV & V can provide an independent assessment of development
  - Is program on schedule, within budget?
  - Have all requirements been correctly implemented?
  - Can design trace back to requirements?
  - Is test plan comprehensive?
  - Is configuration management effective?

- IV & V is not always required for:
  - Developers with mature processes
  - Low risk system or subsystem

- Downside: IV & V can be costly
  - Must weigh costs against expected benefits in quality
  - Typically “earlier is better” in terms of cost-effectiveness
COTS Software Issues & Risks

- Unit level testing/component testing is generally impossible
- Incomplete documentation
- Multiple complex, non-standard interfaces
- Market leverage may not exist to force vendor bug fixes
- Formal requirements documents unavailable
- COTS usage may not match original design environment

- Real-time performance may be marginal
- Robustness & reliability lower when compared to custom code
- Higher COTS use in a system generally implies more difficult system level integration testing
- Frequent market-driven releases complicate regression testing
Chart illustrates that maintenance costs increase steeply as the number of COTS components increase

Source: Added Sources of Costs in Maintaining COTS-Intensive Systems, Clark, CrossTalk, June 2007
Software Issues (prior to Operational Test & Evaluation)

- Does the software support system performance?
- Is the software mature & reliable?
- Is the software usable by typical operators?
- Is the software sustainable at a reasonable cost?

Software Maturity or completeness “should be demonstrated prior to OT&E”
Software Testing Lessons “Learned”

• Formulate test strategy *prior* to contract award that accommodates cost/schedule constraints

• Test Strategy should be able to:
  – Verify all critical software requirements of system
  – Test in a way to isolate faults

• Phase testing to focus on:
  – Qualification Testing
  – Operational Thread Testing
  – Performance/Stress Testing

• Resolve the “requirements” vs. “design” information argument early-on
Software Test and Evaluation Purposes

- Clearly defined risk assessment criteria for each life cycle phase
- Must provide data to support qualitative & quantitative metrics
- Incremental test strategy
  - Identify & correct errors early
- Provide data to support acquisition decisions
Exercise # 2

Based on your own expertise, experiences, or ‘tales of woe and/or joy’ (lessons learned perhaps), is there really a universal set of ‘best/better practices’ for software? If so, ……

1. Describe a better practice.
2. Why is it a better practice?
3. If implemented, what are the implications?
4. Describe a ‘worst practice’.
Quality is the result of conscious, careful plans for quality processes and practices.
Project Integrity

• Adopt Continuous Program Risk Management
  – Risk management is a continuous process beginning with concept definition

• Estimate Cost and Schedule Empirically
  – Initial software estimates and schedules should be looked on as high risk

• Use Metrics to Manage
  – All programs should have a metrics program in place to monitor issues

• Track Earned Value
  – EV project management requires a WBS, work packages, activity networks at every WBS level, accurate estimates, and implementation of a consistent/planned process.

• Track Defects against Quality Targets
  – All programs need to have pre-negotiated quality targets

• Treat People-as the Most Important Resource
  – A primary program focus should be staffing positions with qualified personnel and retaining this staff through the life of the project.
• **Adopt Life Cycle Configuration Management (CM)**
  – All programs, irrespective of size, need to manage information through a preplanned configuration management process.

• **Manage and Trace Requirements**
  – Before any design is initiated, requirements for that segment of the software need to be agreed to.

• **Use System-Based Software Design**
  – All methods used to define system architecture and software design should be documented in the system engineering management plan and software development plan and be frequently and regularly evaluated through audits conducted by an independent program organization.
Construction Integrity (continued)

• Ensure Data and Database Interoperability
  – All data and database implementation decisions should consider interoperability issues and, as interoperability factors change, these decisions should be revisited.

• Define and Control Interfaces
  – Before completion of system-level requirements, a complete inventory of all external interfaces needs to be completed.

• Design Twice, Code Once
  – All design processes should follow methods documented in the software development plan.

• Assess Reuse Risks and Costs
  – The use of COTS, reuse, or any other non-developmental items should be treated as a risk and managed accordingly through risk management.
• **Inspect Requirements and Design**
  – All products that are placed under CM and are used as a basis for subsequent development need to be subjected to successful completion of a formal inspection prior to its release to CM.
  – Goal of 80% defect detection

• **Manage Testing as a Continuous Process**
  – All testing should follow a preplanned process, which is agreed to and funded.

• **Compile and Smoke Test Frequently**
  – All tests should use systems that are built on a frequent and regular basis (nominally no less than twice a week).
  – All new releases should be regression tested prior to release to the testing organization.
Summary

- Software quality and quality issues unique to software
- Processes and practices associated with quality software development efforts
- Software quality initiatives, policies and documentation
- Software testing best practices and types of software tests
- Software testing including independent verification and validation (IV&V) throughout the acquisition lifecycle
References

The Data and Analysis Center for Software (DACS)
https://www.thedacs.com/

Practical Software and Systems Measurement (PSM)
http://www.psmsc.com

Software Engineering Information Repository (SEIR)
https://seir.sei.cmu.edu/seir/

Software Program Manager’s Network (SPMN)
http://www.spmn.com/lessons.html

Software Engineering Institute (SEI) – Carnegie Melon
http://www.sei.cmu.edu/

DoD Information Technology Standards Registry (DISR Online)
https://disronline.disa.mil/a/public/index.jsp

Contact Information

Larry Baker
Defense Acquisition University
(703) 805-3636
larry.baker@dau.mil