Incorporating Quality Throughout the Lifecycle

Presented by:
Betty Schaar
BenchmarkQA

Date:
October 2, 2008
BenchmarkQA

• **Software Quality Assurance Consulting and Staffing**
  – Specialized consulting in:
    • QA and testing process assessment and improvement
    • Test automation
    • Performance engineering
  – Staff augmentation and direct placement of true testing and quality assurance professionals with all levels of expertise.
  – More than 20 years of integrity and service excellence, resulting in loyal and satisfied clients
  – Extensive expertise in complex system conversion and data testing.

• **Local Test Lab**
  – Over 15 years of experience bringing projects into our lab (Microsoft, IBM, Tyco, Avery).
  – Proprietary processes and methods provide thorough testing and effective project management.

• **Training**
  – Public and private training to enhance QA processes and management.
  – Customized programs facilitate large team ramp-up and enhance consistency.
  – **Structured Software Test Planning, November 6 & 7th**, Edina
Betty Schaar

• As a senior consultant for BenchmarkQA, Betty delivers process improvement and training services.
• A testing and QA professional for more than 20 years.
• Certified Software Quality Analyst (CSQA) through Quality Assurance Institute.
• QA leadership and consulting roles at United Healthcare, Blue Cross Blue Shield, Kodak Health Imaging, Imation and the State of Minnesota.
• QA expertise highlighted through speaking engagements at PSQT and STAR conferences, and leadership in TCQAA and PSQT.
Agenda

• Costs of Quality – NIST Study
• Rework Costs
• Relative Cost of Finding and Fixing Defects
• Prevent Defects to Lower Project Costs
• Determine Your Costs to Find and Fix Defects
• Project Estimations and Better Approaches
• Elements of a Good Testing Process
• Creating an Action Plan
“Quality is never an accident, it is always the result of high intention, intelligent direction and skillful execution; it represents the wise choice of many alternatives.”

— William A. Foster
Costs of Quality
– NIST Study
NIST Study in 2002
The Economic Impacts of Inadequate Infrastructure for Software Testing

National Annual Costs of an Inadequate Infrastructure for Software Testing Realized from Defects:

$59.5 BILLION DOLLARS (estimated)

$38.3 Billion is attributed to error avoidance and mitigation activities required by software users.

$21.2 Billion is attributed to rework costs by project teams.
NIST Study in 2002 - Cont’d

Potential Cost Reduction from Feasible Infrastructure Improvements:

$22.2 BILLION DOLLARS (estimated)

$11.7 Billion for software users.

$10.6 Billion for project teams.
### NIST Study in 2002 - Cont’d

**Defect Fix Time Based on Discovery Point**

<table>
<thead>
<tr>
<th>Financial Services Sector</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>1.2</td>
</tr>
<tr>
<td>Coding/Unit Testing</td>
<td>4.9</td>
</tr>
<tr>
<td>Integration</td>
<td>9.5</td>
</tr>
<tr>
<td>Beta Testing</td>
<td>12.1</td>
</tr>
<tr>
<td>Post-Product Release</td>
<td>15.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transportation Mfg Sector</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>2.0</td>
</tr>
<tr>
<td>Coding/Unit Testing</td>
<td>2.4</td>
</tr>
<tr>
<td>Integration</td>
<td>4.1</td>
</tr>
<tr>
<td>Beta Testing</td>
<td>6.2</td>
</tr>
<tr>
<td>Post-Product Release</td>
<td>13.1</td>
</tr>
</tbody>
</table>

Note that the NIST document was not clear regarding which activities were/were not included in these times. These numbers may actually be low if not inclusive of all defect find and fix related activities.
“If we are busy doing rework for defects, we’re not innovating AND we are costing the company lots of money.”

— Anonymous
Rework Costs
Rework Costs

Rework costs can run as high as 50% or more of a project’s total costs

• If you haven’t looked at your actual rework costs, chances are good you underestimate them.
• Rework doesn’t create revenue or provide new capabilities; new development does.
Rework Costs - Cont’d

Lack of Good Requirements*

= Assumptions

= Rework

*Due to lack of user involvement to define requirements, poorly written requirements, lack of requirements reviews to clarify needs
“It is much less expensive to prevent errors than to rework, scrap, or service them.”

— Philip Crosby
Relative Costs of Finding and Fixing Defects
Relative Cost of Finding and Fixing a Defect Across the Project Lifecycle

45% of all errors are analysis errors. 35% of all errors are design errors. Only 20% of errors originate during coding.

“Pay me now or pay me (more) later…”

— *Testers Creed*
Prevent Defects to Lower Project Costs
Fact: Bugs Beget Bugs

- Prevent “Bug X” (that originated in requirements or design) from getting into the code means rework costs are not incurred for “Bug X”
- Plus other bugs (and associated costs) that might have been introduced when “Bug X” was fixed are also prevented
Fact: Break/Fix Cycles are Disruptive, Difficult to Plan

- Forward progress is slowed or has to stop while a defect is researched and a decision made to fix or defer
- Best attempts to plan a software project can only guess at number of break/fix cycles needed
- If more break/fix cycles are needed than planned, one of the following typically happens:
  1) Something doesn’t get adequately tested; quality ramifications
  2) Timelines are extended; monetary ramifications
  3) Both 1 and 2
Fact: Break/Fix Cycles are Typically Done in a Rush

- Defects found during testing phase result in a rushed effort on everyone’s part to react to problems; something will be missed and perhaps will not be found until the software is released.

- Discovery of a defect after production release can have serious monetary, legal, and/or customer satisfaction ramifications.
What Is Static Testing?

Definition:
- Analysis of a project deliverable or program logic without executing the program.
- Inspecting and reviewing document work products that are used to define and design the software.

As compared to Dynamic Testing which is:
- Testing software by executing it or interacting with it.
# Requirements Reviews

<table>
<thead>
<tr>
<th>Questions to Ask:</th>
<th>Development</th>
<th>SQA / Test Team</th>
<th>Technical Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can the requirement be implemented?</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is the requirement testable?</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Is the requirement unambiguous?</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>- Am I making any assumptions?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Are there any conflicting requirements?</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Are there any redundant requirements?</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Are there any missing requirements?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Have terms been used consistently?</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Have the requirements standards been followed?</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>What are the implicit requirements?</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
## Design Reviews

<table>
<thead>
<tr>
<th>Questions to Ask:</th>
<th>SQA / Test Team</th>
<th>Technical Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Are there any conflicts between design and requirements?</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Are there any requirements not addressed in the design?</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>What are the system and data interfaces and handoff points?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Is there any complex logic?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Are there any GUI layouts?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Are there any file layouts?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Are there any report layouts?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Is there a database schema/specifications, and/or data rules?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>What other functional rules and logic should be considered for testing or documents?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Have terms been used consistently?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Have the design standards been followed?</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Typical Approach to Project Cycles

If your project cycles look like this, something needs to change or one or all of the following will occur:

- Cost to find/fix defects is going to be far greater than needs to be
- Likelihood of missed deadlines is high due to late discovery of problems
- Likelihood for project cost overruns is high due to rework for late discovery of problems
Ideal Approach to Project Cycles

Adjusting how releases/iterations are scheduled as shown below results in test team involvement in requirements definition and document reviews, thereby enabling static testing.
Efficient Approach to Project Cycles

Adjusting how releases/iterations are scheduled as shown below results in better utilization of all project resources while also enabling the test team to be involved in document reviews, as they do not occur during test phase.
Other Ideas for Enabling Static Testing in the Project Cycle

“Leap Frog” Primary and Secondary Resources

Person/Team 1:  
– Primary for Release/Iteration 1, 3, etc.
– Secondary for Release/Iteration 2, 4, etc.

Person/Team 2:  
– Primary for Release/Iteration 2, 4, etc.
– Secondary for Release/Iteration 1, 3, etc.

Primary Responsibilities (testing):
• Participate in Document Reviews
• Write the Test Plan
• Test Case Design
• Test Execution
• Status and Metric Reporting

Secondary Responsibilities (testing):
• Assist with Test Case Design
• Assist with Test Execution
Other Ideas for Enabling Static Testing in the Project Cycle - cont’d

- Designate one test resource as static tester for all releases/iterations
  - Based on who is best at it; supports other testing activities only as time permits

- Static testing is job function of Sr. Test Analyst or Test Lead role only
  - Have junior test resources focused on test execution cycles

- Plan extra time into Release/Iteration 1 Test Phase
  - To review requirements and/or design documents and attend review meetings for Release/Iteration 2
  - Minimum of 2 day lead time ahead of review meeting, for adequate review of documents
“The probability of the existence of more errors in a section of a program is proportional to the number of errors already found and fixed in the section.”

— A Testing Principle from Glenford Myers
Determine Your Costs to Find and Fix Defects
Root Cause Tracking & Assessment

- Assign standard root cause value to each defect record
- Use the “5 Whys” method to get to true root cause
  - Not just symptoms and not root fix!
- Assess root cause classifications as a project team during post-mortem/retrospective, change classifications as needed
Using Root Cause Data For Improvements

• Project teams look for project trends
  – Address with corrective action(s) for their projects

• QA organization looks for corporate trends
  – Address with organization-wide process changes
Defect Find/Fix Cost Factors

Pre-release

Include time for all of the following activities:

- Test execution to uncover the defect
- Proving if defect is reproducible
- Record defect
- Evaluation - work now or defer?
- Analysis and research the design and code
- Fixing the code
- Build, unit test, any additional work on the code
- Update defect record with resolution
- Integration and system test for fix
- Regression test around the fix
- Defect closure
Defect Find/Fix Cost Factors

Post-release

Include time for most, if not all, pre-release activities PLUS:

- Determination of work-around (if applicable)
- Communication to all impacted parties
- Updating documentation (if applicable)
- Production code release
- Production validation
- Data clean-up (if applicable)
- Others?
Calculating Find/Fix Cost Per Defect

Per defect find/fix cost, for defects both found and fixed during *system test phase*:

<table>
<thead>
<tr>
<th>Average number of hours to find/fix a defect</th>
</tr>
</thead>
<tbody>
<tr>
<td>(time for all team members involved, added together)</td>
</tr>
</tbody>
</table>

\[ \times \text{Average burden rate} \]

(salary + benefits, a blended average for combined job levels)

e.g. 12 hours x $75/hour = $900.00 per defect!
Calculating Total Phase Find/Fix Rework Costs

Total costs for all defects found/fixed during system test phase:

- Average number of hours to find/fix a defect
  (time for all team members involved, added together)

- $ \times $ Average burden rate
  (salary + benefits, a blended average for combined job levels)

- $ \times $ Total number of defects found and fixed during phase

\[ \text{e.g. 12 hours} \times \$75/\text{hour} \times 30 = \$27,000.00 \text{ of rework costs in just the system test phase!} \]
Calculating Total Project Find/Fix Rework Costs

Total costs for all defects found/fixed during all pre-release phases:

- Total costs to find/fix defects in Phase 1/Requirements
- Total costs to find/fix defects in Phase 2/Design, Code
- Total costs to find/fix defects in Phase 3/System Test
- Total costs to find/fix defects in Phase 4/Accept. Test

E.g. $3,000 + $10,000 + $27,000 + $60,000 = $100,000 of rework costs pre-release!
“You can't manage what you can't control, and you can't control what you don't measure.”

— Tom DeMarco
Project Estimations
and
Better Approaches
Common Estimation “Rules of Thumb”
Tester-to-Developer Ratio

<table>
<thead>
<tr>
<th>Organization</th>
<th>Leading Edge</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test</td>
<td>Devel.</td>
</tr>
<tr>
<td>Internal IS</td>
<td>1</td>
<td>2.5 - 4</td>
</tr>
<tr>
<td>Software Product Vendor</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Software Services Vendor</td>
<td>1</td>
<td>2 - 3</td>
</tr>
</tbody>
</table>
Problems with Tester-to-Developer Ratios

• Does not account for actual duties to be performed.
• What roles count as “Testers”? As “Developers”? 
• Is this the same project-to-project so that we can leverage historical data for estimating purposes?
• What is the experience level of each category?
• New development work might require more developers than testers but maintenance work might require more testers than developers.
• Concept only makes sense for longer range forecasting, not detailed planning.
Common Estimation “Rules of Thumb”
Gross Labor Ratio

For traditional projects – based on 100 hours of coding effort

<table>
<thead>
<tr>
<th>Activity</th>
<th>Hours</th>
<th>% of Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Initiation</td>
<td>25</td>
<td>3.7%</td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td>200</td>
<td>29.6%</td>
</tr>
<tr>
<td>Coding</td>
<td>100</td>
<td>14.8%</td>
</tr>
<tr>
<td>Unit Testing</td>
<td>100</td>
<td>14.8%</td>
</tr>
<tr>
<td>Integration &amp; System Test</td>
<td>200</td>
<td>29.6%</td>
</tr>
<tr>
<td>Installation / Deployment</td>
<td>50</td>
<td>7.9%</td>
</tr>
<tr>
<td>TOTAL</td>
<td>675</td>
<td>100.00</td>
</tr>
</tbody>
</table>
Problems with Gross Labor Ratios

• Only applicable for new development projects

• For maintenance projects testing might need to be 100% or more of development as regression testing is not proportional to design and coding.

• Concept only makes sense for longer range forecasting, not detailed planning
Better Approaches to Estimation

• Estimates based on sized requirements
  – T-shirt method of sizing - XS, S, M, L, XL
    Each size has an associated average time factor or each functional team has own time factor for each size of requirement

• Estimates based on function points
  – www.ifpug.com
  – Weighted totals of five external aspects of software applications:
    • Types of inputs to the application
    • Types of outputs from the application
    • Types of inquiries users can make
    • Types of logical files which the application maintains
    • Types of interfaces to other applications
“Definition - Insanity: Doing the same thing over and over, and expecting different results.”

— Albert Einstein
Elements of a Good Testing Process
The BenchmarkQA Structured Software Testing Process

- Organization Testing Policy/Charter
- Requirements
  - Test Strategy/Test Plan
- Test Requirements/Traceability Matrix
- Test Cases/Checklists/Scenarios/Scripts
- Unit Tests
- Integration Tests
- System Tests
- Acceptance/Beta Tests
- Test Data

©2008 BenchmarkQA
## Testing Approaches

Include all Three in Your Projects

<table>
<thead>
<tr>
<th>Testing Approach</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Box Testing</td>
<td>Testing without knowledge of the inner workings of the software; the focus is on application or system inputs and outputs.</td>
</tr>
<tr>
<td>White Box Testing</td>
<td>Testing based on software structure, logic paths, and methods; is also known as Clearbox or Glassbox.</td>
</tr>
<tr>
<td>Gray Box Testing</td>
<td>Testing that is a blend of Black box and White box approaches; typically this means the system or application design is taken into account (as defined in a design document) when designing tests.</td>
</tr>
<tr>
<td>Unit Test</td>
<td>Integration Test</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Objectives</strong></td>
<td>Verify individual functions, modules, or units of code are working correctly</td>
</tr>
<tr>
<td>Test Approach</td>
<td>White Box</td>
</tr>
<tr>
<td><strong>Accountable Roles</strong></td>
<td>Development Team: Developer, Technical Lead</td>
</tr>
<tr>
<td>Responsibilities</td>
<td>Write software code, Write, execute unit tests, Contribute to master test plan or write unit test plan, Review testing documents for next test phases, Resolve defects, Support next test phases</td>
</tr>
<tr>
<td><strong>Typical Entry Criteria</strong></td>
<td>Requirements are defined, Design is complete, Code is written, Unit tests are defined, At least two interfacing units / modules of code have been unit tested and are checked in to the version control system, Integration tests are defined</td>
</tr>
<tr>
<td><strong>Typical Exit Criteria</strong></td>
<td>Unit tests pass, Integration tests pass</td>
</tr>
</tbody>
</table>
Identify Test Requirements

Look for requirements in the following sources:

- Requirements documents
- Design documents
- Organization standards
- Regulations
- As-built systems

Identify both explicit and implicit requirements to be tested!
Use Smart Test Design Techniques

- Equivalence Partitioning
- Limits/Boundary Testing (Boundary Plus/Minus One)
- Cause-Effect Graphing
- Decision Tables

Include both positive and negative tests!
Effective and Efficient Test Design

• Risk rankings
  – Assigned to each test requirement and focus efforts around higher risk items first (aka risk-based testing)

• Define actions, expected results for max flexibility
  – e.g. reference test data rather than embedding records into the detailed tests

• Repeatable tests
  – Write detailed tests that are repeatable by anyone on test team

• Requirements coverage
  – At least one detailed test for each test requirement
  – Each detailed test satisfies a very limited number of test requirements (ideally just one)
Effective and Efficient Test Design – cont’d

• Common tests
  – For recurring processes or interface elements

• Test checklists
  – For verifying standards – GUI controls, report layouts, etc.

• Modular tests
  – That can be called/chained together with other tests rather than writing really long tests

• Testing standards that include:
  – Consistent formats and templates
  – Conventions for test naming, actions to take in tests, test data, how application objects are referred to
Test Automation

- Not every test is a good candidate for automation. Consider automating the following:
  - Tests that will be repeated three or more times with the same application functionality (e.g. “smoke test”)
  - Data input time is significant – can instead loop through a file of values with a script
  - Test executor boredom or tedium could reduce the effectiveness of the test
  - Performance and stress tests
  - As much of the regression suite as is possible

- Use programming guidelines for scripting
- Test the tests!
Peer Reviews of Testing Documents

- Allow test artifact to be in best possible state before broader distribution
- Helps to maintain consistency in content of testing artifacts
- Identify deviations from standards
- Provide additional perspectives and input on what and how to test
- Promote understanding of projects and technologies across team members
Formal Reviews of Testing Documents

• Get agreement on the planned testing and collect signatures to show approval

• Test Plan review usually includes project stakeholders from business/user community and from project team
  – Helpful to include first draft of Traceability Matrix, outlining test requirements

• Detailed test review might include project manager, technical lead and/or developer, business analyst or other business representative
“Since testing is a costly activity, one wants to recover some of its cost by increasing the worth of the program.”

— Glenford Myers
Creating an Action Plan
Creating an Action Plan

The key steps to making process improvement changes are:

1. Establish ‘true’ root cause tracking and monitoring
2. Determine your organization’s cost of defects
3. Implement changes to lower overall costs of defects
‘True’ Root Cause

If root cause is currently tracked:

- What classifications are used?
  - Are they sufficient?
- Is root cause a required element in the defect tracking system?
- What job role enters root cause initially?
  - Is this role choice acceptable to all?
- Is the root cause classification for each defect reviewed by the project team at the close of a release?
‘True’ Root Cause

If root cause is not currently tracked:

- Establish the set of root cause choices. Suggestions:

<table>
<thead>
<tr>
<th>Requirements – Missed Item</th>
<th>Code – Missed Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements – Incorrect</td>
<td>Code – Bug</td>
</tr>
<tr>
<td>Requirements – Conflict</td>
<td>Code – Not Supporting Design</td>
</tr>
<tr>
<td>Design – Missed Item</td>
<td>Code – Not Supporting Requirements</td>
</tr>
<tr>
<td>Design – Error</td>
<td>Test Case Error</td>
</tr>
<tr>
<td>Design – Not Supporting Requirements</td>
<td>Documentation Error</td>
</tr>
</tbody>
</table>

- Modify the defect tracking tool
  - Who and when?
  - What to do with existing defects – what value to assign?
Determine Your Costs of Defects

• What project life cycle phases are used consistently?
• What activities should time be tracked in each phase that are relevant to finding and fixing a defect?
• Target a variety of projects and a handful of defects from each to collect the time details
  – Include various sized projects
• This data can be useful in garnering support for additional resources, process improvements, tools, etc. and to know how much time and energy should be spent based on the anticipated return on investment
Implement Changes

• Hold a cross-functional brainstorming discussion on what could/should be changed that would help to lower overall costs of defects
• After ideas are exhausted, group and rank them by priority
• Look for volunteers to work on championing and piloting a small number of process changes
• Evaluate the effectiveness of the changes
  – Adjust processes as appropriate
• Continue to work on improvements for other problematic areas
“Failure is an opportunity to start again more intelligently.”

— Henry Ford
Recommended Reading


Implementing Lean Software Development – From Concept to Cash, Mary and Tom Poppendieck. Addison-Wesley Professional, 2006

Principles of Quality Costs, Jack Campanella, ASQ Quality Press, 1999

Quality is Free, Philip Crosby, McGraw-Hill, 1979

Prepared by RTI for NIST, 2002

Khaled El Emam PhD, 2003
Helpful Website References

• http://www.sei.cmu.edu
• http://www.stickyminds.com
• http://www.leansoftwaredevelopment.org/
• http://www.agilealliance.org/
• http://www.deming.org/
• http://www.soft.com/Institute/HotList/index.html
• http://www.asq.org/
• http://www.rti.org/publications.cfm?nav=364
• http://www.ifpug.org/
• http://www.spr.com/
Thank You!

For additional information on how BenchmarkQA can assist with improving your QA and testing function, contact:

Molly Decklever
952.392.2384
Molly.Decklever@benchmarkqa.com