Predicting Software Quality Early in the Software Development Lifecycle and Producing Secure Software

Girish Seshagiri

ASQ Software SIG
October 25, 2011
Main Points - 1

Defective software is insecure

Managing the software work is difficult because of the way it is currently done

To manage software, you need to manage quality

To manage software quality, developers need to manage quality of the software components built by them
Main Points - 2

Procedural processes do not change engineering behavior

Need operational processes at the team and individual levels

Performance metrics that really matter come from the developers themselves and provide early warning
Why Are We Here? - 1

The adverse impact of software vulnerabilities caused by defective software is far-reaching.

The defects that escape testing are exploited by hackers to launch cyber attacks.

The current method of dealing with the increasing number of cyber attacks is reactive.
Why Are We Here? - 2

We need a rapid transformation of the U.S. software industry from the current “Deliver now, fix later” culture to one capable of delivering substantially defect free code within predictable cost and schedule.

This is a national high-priority need.

If we continue with current methods, the U.S. taxpayer will pay billions of dollars for fixing defects in delivered products.
Why are We Here? - 3

<table>
<thead>
<tr>
<th></th>
<th>Instructor Estimate</th>
<th>Your Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated size of modernization software in Function Points</td>
<td>20,000,000</td>
<td>A</td>
</tr>
<tr>
<td>Estimated defect potential per Function Point at current capability</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>Defect potential</td>
<td>100,000,000</td>
<td>0</td>
</tr>
<tr>
<td>% Cumulative defect removal efficiency</td>
<td>85</td>
<td>D</td>
</tr>
<tr>
<td>% Defects in delivered product</td>
<td>15</td>
<td>E=100-D</td>
</tr>
<tr>
<td>No. of defects in delivered product</td>
<td>15,000,000</td>
<td>F=E/100*C</td>
</tr>
<tr>
<td>Cost to fix a defect</td>
<td>4,000</td>
<td>G</td>
</tr>
<tr>
<td>Total cost to taxpayer</td>
<td>$60,000,000,000</td>
<td>$0</td>
</tr>
</tbody>
</table>

Estimated cost to taxpayer due to poor quality of delivered software

Instructor Estimate Your Estimate

<table>
<thead>
<tr>
<th></th>
<th>Instructor Estimate</th>
<th>Your Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated size of modernization software in Function Points</td>
<td>20,000,000</td>
<td>A</td>
</tr>
<tr>
<td>Estimated defect potential per Function Point at current capability</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>Defect potential</td>
<td>100,000,000</td>
<td>0</td>
</tr>
<tr>
<td>% Cumulative defect removal efficiency</td>
<td>85</td>
<td>D</td>
</tr>
<tr>
<td>% Defects in delivered product</td>
<td>15</td>
<td>E=100-D</td>
</tr>
<tr>
<td>No. of defects in delivered product</td>
<td>15,000,000</td>
<td>F=E/100*C</td>
</tr>
<tr>
<td>Cost to fix a defect</td>
<td>4,000</td>
<td>G</td>
</tr>
<tr>
<td>Total cost to taxpayer</td>
<td>$60,000,000,000</td>
<td>$0</td>
</tr>
</tbody>
</table>
Managing the Software Work
State of the Practice - 1

In spite of increased adoption of PMP, Six Sigma, Lean, Agile, CMMI, ITIL etc, industry performance outcomes for cost, schedule and quality continue to be mixed

Changing managers, procurement regulations, acquisition procedures, and contracting provisions including oversight reporting have not resolved the cost, schedule, and quality problems of most software projects

Even with all of the emphasis on writing software with security in mind, most software applications remain riddled with security holes
Managing the Software Work
State of the Practice - 2

Current software quality methods rely on testing, and are expensive, unpredictable and ineffective.

Heroic efforts rescue troubled projects; heroes are in short supply.

High-performance teams are the exceptions and not the norm.
Common Misconceptions About The Software Work - 1

We must start with firm requirements
If it passes test, it must be OK
Software quality can’t be measured
The problems are technical
We need better people
Software management is different

Source: Managing the Software Process, Watts Humphrey
Common Misconceptions About The Software Work – 2

Maturity level 3 is all that is needed
Higher maturity levels add to cost
Higher maturity levels are needed only for safety critical or business mission critical systems
If it is “agile” or “lean”, it is good
What we need are lean processes
Maturity levels guarantee results
Maturity level 5 is the end
Contracts and Management Systems - 1

Too often, our contracts and management systems are based on the common misconceptions

Time and Materials and Cost Plus Fee contracts end up rewarding poor cost, schedule and quality performance

CMMI levels 2 or 3 specified as pass/fail criteria

Emphasis on product technical reviews which are important, but do not motivate change and are not helpful to improve the process or track status
Contracts and Management Systems - 2

Not trusting teams to estimate the job, prepare aggressive and realistic plans, and negotiate responsible commitments

Lack of precision and timeliness in reporting contract performance and job status

Multi-contractor teams lack sense of purpose, goals and mission and do not act as a real team
Current Methods
“Deliver Now, Fix Later”

Command and control management system not suitable for managing knowledge work
Workers and managers have different objectives and different views of project success
Lack of trust between management and teams
Teams unable to negotiate schedule and cost commitment
Arbitrary, unrealistic schedules creating culture of “Deliver now, fix later”
Lack of precise project status measures
The Top 15 U.S. Software Cost Drivers in Rank Order Circa 2009

1. The cost of finding and fixing bugs
2. The cost of cancelled projects
3. The cost of producing paper documents and English words
4. The cost of recovery from security flaws and attacks
5. The cost of requirements changes during development
6. The cost of programming or coding
7. The cost of customer support
8. The cost of meetings and communication
9. The cost of project management
10. The cost of application renovation
11. **The cost of innovation and new kinds of features**
12. The cost of litigation for cancelled projects
13. The cost of training and learning software applications
14. The cost of avoiding security flaws
15. **The cost of acquiring reusable components**

Source: Capers Jones, STN 13-1 April 2010: Software Quality, Reliability, and Error Prediction
# Schedule Compression and Quality

<table>
<thead>
<tr>
<th>Schedule/Quality Trade-off</th>
<th>Default</th>
<th>10% Compression</th>
<th>20% Compression</th>
<th>10% Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration Mths</td>
<td>25.9</td>
<td>23.3</td>
<td>20.7</td>
<td>28.5</td>
</tr>
<tr>
<td>Defect Count</td>
<td>1,033</td>
<td>1,316</td>
<td>1,715</td>
<td>849</td>
</tr>
<tr>
<td>% Change</td>
<td></td>
<td>27.4%</td>
<td>66.0%</td>
<td>-17.8%</td>
</tr>
</tbody>
</table>

Source: Donald M. Beckett and Douglas T. Putnam, STN 13-1 April 2010: Software Quality, Reliability, and Error Prediction
Adding Staff and Quality

<table>
<thead>
<tr>
<th></th>
<th>Peak Staff 16</th>
<th>Peak Staff 32</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration Mths</td>
<td>26</td>
<td>22.6</td>
<td>-13.1%</td>
</tr>
<tr>
<td>Defect Count</td>
<td>1,043</td>
<td>1,411</td>
<td>35.3%</td>
</tr>
<tr>
<td>Effort Months</td>
<td>225</td>
<td>392.0</td>
<td>74.2%</td>
</tr>
</tbody>
</table>

Source: Donald M. Beckett and Douglas T. Putnam, STN 13-1 April 2010: Software Quality, Reliability, and Error Prediction
## Defect Injection Rate and Removal Percent - Sample

<table>
<thead>
<tr>
<th>Phase</th>
<th>Injection Rate – Defects Per Hour</th>
<th>Percent Defects Removed - Yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>High Level Design</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>Detailed Design</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>Unit Test</td>
<td></td>
<td>50%</td>
</tr>
<tr>
<td>Integration Test</td>
<td></td>
<td>55%</td>
</tr>
<tr>
<td>System Test</td>
<td></td>
<td>35%</td>
</tr>
</tbody>
</table>

Your organization/projects have such data?
## Defects Per Function Point

<table>
<thead>
<tr>
<th>Defect Origins</th>
<th>Defect Potentials</th>
<th>Removal Efficiency</th>
<th>Delivered Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>1.00</td>
<td>77%</td>
<td>0.23</td>
</tr>
<tr>
<td>Design</td>
<td>1.25</td>
<td>85%</td>
<td>0.19</td>
</tr>
<tr>
<td>Coding</td>
<td>1.75</td>
<td>95%</td>
<td>0.09</td>
</tr>
<tr>
<td>Document</td>
<td>0.60</td>
<td>80%</td>
<td>0.12</td>
</tr>
<tr>
<td>Bad Fixes</td>
<td>0.40</td>
<td>70%</td>
<td>0.12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>5.00</strong></td>
<td><strong>85%</strong></td>
<td><strong>0.75</strong></td>
</tr>
</tbody>
</table>

Source: *Capers Jones, STN 13-1 April 2010: Software Quality, and Software Economics*
Managing Software Quality - 1

To meet schedule and cost commitments consistently, you must manage software quality.

Quality without numbers is just talk.

The common ways to manage software quality are with testing and reuse.

Testing is now relied upon and is not sufficient.

For reuse, the parts must be initially of high quality or the quality problems will be worse.
Managing Software Quality - 2

Software-intensive products typically have many defects

The three defect removal strategies
  Test, test, test
  Inspect and test
  Review, inspect and test

Time to find and fix test defects can vary from a few hours to few weeks
Managing Software Quality - 3

Quality work is more predictable

If you do not manage quality, your schedule problems will end up as quality disasters

Software professionals must be trained to make plans and negotiate commitments
CMMI Improves Quality - 1

<table>
<thead>
<tr>
<th>CMM Level</th>
<th>Defects/KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>7.5</td>
</tr>
<tr>
<td>Level 2</td>
<td>6.24</td>
</tr>
<tr>
<td>Level 3</td>
<td>4.73</td>
</tr>
<tr>
<td>Level 4</td>
<td>2.28</td>
</tr>
<tr>
<td>Level 5</td>
<td>1.05</td>
</tr>
</tbody>
</table>
CMMI Improves Quality - 2

<table>
<thead>
<tr>
<th>CMM Level</th>
<th>Defect Potential per Function Point</th>
<th>Defect Removal Efficiency</th>
<th>Delivered Defects per Function Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEI CMM 1</td>
<td>5.50</td>
<td>73.00%</td>
<td>1.49</td>
</tr>
<tr>
<td>SEI CMM 2</td>
<td>4.00</td>
<td>90.00%</td>
<td>0.40</td>
</tr>
<tr>
<td>SEI CMM 3</td>
<td>3.00</td>
<td>95.00%</td>
<td>0.15</td>
</tr>
<tr>
<td>SEI CMM 4</td>
<td>2.50</td>
<td>97.00%</td>
<td>0.08</td>
</tr>
<tr>
<td>SEI CMM 5</td>
<td>2.25</td>
<td>98.00%</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Source: Capers Jones, STN 13-1 April 2010: Software Quality, and Software Economics
Issues with Procedural Processes

Procedural processes rely on artifacts to make sure process is being followed

Artifacts are produced by organizational bureaucracy and not the developers

Artifacts may have no relationship to the actual work being done

Easier to pass appraisals than to change engineering behavior
Need for an Operational Process

The need is not for lots of process data and artifacts but for engineers who develop and use artifacts and data.

What would happen if software professionals used sound engineering practices?
- Made and followed detailed plans
- Gathered and used historical data
- Measured and managed quality
- Analyzed and improved their processes

The need is for an operational process at the team and individual levels.
Managing the Software Work - 1

Software and systems development is knowledge work

The first rule for knowledge work is that managers can’t manage it - the workers must manage themselves
The second rule is that developers and their teams
  Must know how to manage themselves
Negotiate their commitments with management
  Manage with data
Own their own work

The third rule is that management must trust the
development teams to plan and manage their own work

Source: Acquiring Quality Software, Watts Humphrey
To manage software product quality, we need to manage quality of the components.

Components are what developers build.

Error-prone components account for a disproportionate number of defects found in integration, system and user acceptance tests.
20% of the modules in a system typically account for 80% of the defects

It is extremely useful to know which components are likely to be error-prone in later testing so that we can take corrective actions pro-actively
Component Development Process

Planned & Actual Object Size Data
Planned & Actual Productivity Data
Planned & Actual Effort Data
Planned Value & Earned Value
Planned & Actual Time Spent in Each Phase
Planned & Actual Defect Data

Produce Conceptual Design

Estimate Objects

Estimate LOC

Estimate Effort

Develop Schedule

Design
Personal Design Review
Team Design Inspection
Code
Personal Code Review
Compile
Team Code Inspection
Unit Test

Develop Product

Collect Process Data

Analyze Process Data
# Performance Metrics That Matter

## Benchmarking

<table>
<thead>
<tr>
<th>Metric</th>
<th>Industry Average</th>
<th>AIS Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule deviation</td>
<td>&gt;50%</td>
<td>&lt;10%</td>
</tr>
<tr>
<td>No. of defects in delivered product 100,000 LOC</td>
<td>&gt;100</td>
<td>&lt;15</td>
</tr>
<tr>
<td>% of design and code inspected</td>
<td>&lt;100</td>
<td>100</td>
</tr>
<tr>
<td>Time to accept 100,000 LOC product</td>
<td>10 Months</td>
<td>5 Weeks</td>
</tr>
<tr>
<td>% of defects removed prior to system test</td>
<td>&lt;60%</td>
<td>&gt;85%</td>
</tr>
<tr>
<td>% of development time fixing system test defects</td>
<td>&gt;33%</td>
<td>&lt;10%</td>
</tr>
<tr>
<td>Cost of quality</td>
<td>&gt;50%</td>
<td>&lt;35%</td>
</tr>
<tr>
<td>Warranty on products</td>
<td>?</td>
<td>Lifetime</td>
</tr>
</tbody>
</table>
## Performance Metrics That Matter

### Team Goals for Phase Yields

<table>
<thead>
<tr>
<th>Phase Yields (% defect removed)</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Inspections</td>
<td>70%</td>
</tr>
<tr>
<td>Design reviews and inspections</td>
<td>70%</td>
</tr>
<tr>
<td>Code reviews and inspections</td>
<td>70%</td>
</tr>
<tr>
<td>Compiling</td>
<td>50%</td>
</tr>
<tr>
<td>Unit test at 5 or less defects/KLOC</td>
<td>90%</td>
</tr>
<tr>
<td>Integration and system test at &lt;1.0 defects/KLOC</td>
<td>80%</td>
</tr>
<tr>
<td>Before compile</td>
<td>&gt;75%</td>
</tr>
<tr>
<td>Before unit test</td>
<td>&gt;85%</td>
</tr>
<tr>
<td>Before integration test</td>
<td>&gt;97.5%</td>
</tr>
<tr>
<td>Before system test</td>
<td>&gt;99%</td>
</tr>
</tbody>
</table>
Performance Metrics That Matter Early Defect Removal

Developers measure and manage the quality of their individual components and remove defects early through personal reviews and team inspections.

Include OWASP Top Ten and CWE/SANS Top 25 Most Dangerous Programming Errors in review and inspection checklists.

Developers strive to get the highest quality product into test.
Performance Metrics That Matter
Defects Removed by Phase

The project Injected and Removed a total of 1112 code defects throughout the project lifecycle.

- **44.6%** of the defects were injected in Detailed Design
- **46.2%** of the defects were injected in Code
- **90.8%** of total code defects were injected Design & Code
Performance Metrics That Matter
Process Quality Index

PQI is a leading indicator of overall product quality

PQI gives ability to predict whether components that have been unit tested will have down-stream defects in integration, system, and user acceptance testing

Teams can take corrective action and reduce test and rework time
Performance Metrics That Matter
Quality Profile

The above chart shows our 5 measures to achieve our quality goals:

1) Design time to Code time comparison
2) Design Review Time as a % of design time (should be 50% or greater)
3) Code Review Time as a % of code time (should be 50% or greater)
4) Compile Defects per KLOC (Compile should find less than 10 defects per KLOC)
5) Unit Test Defects per KLOC (UT should find less than 5 defects per KLOC)
Performance Metrics That Matter
Percent Defect Free Components

When knowledge workers have been trained and know how to manage themselves, they are capable of giving early warning to management when problems arise.

A key metric is the percent of components in the product that are defect free.

Management can motivate the technical teams to strive for 100% defect free components when both parties view project success the same way.
Microsoft Fixed Schedule Project - 1

Teams negotiated features and resources to fit the schedule

Team members were empowered to stop a project from going into Code, SIT, UAT or Production

Predicted software quality early using metrics

Achieved work life balance

Developers released early from projects

Low budget for post production defects
Microsoft Fixed Schedule Project - 2

Cumulative Earned Value

- Cumulative Planned Value
- Cumulative EV
- Cumulative Predicted Earned Value

Able to React Early!
Microsoft Fixed Schedule Project - 3

Defects Removed by Phase for Assembly SYSTEM

- Plan
- Actual

Phase
- REQ Inspection
- DLD Review
- Code
- Compile
- Unit Test
- System Test

Defects Removed by Phase
Microsoft Fixed Schedule Project - 4

Planned correctly
- Executed correctly
- Planned & Actual PQI > 0.7
- Zero system test defects

Source: Adopting TSP to Fixed Schedule Projects, Microsoft Presentation, TSP Users Group Meeting 2005, Chennai, India
Intuit

Reduced project risk by finding 86% of defects before System Testing even started

Engineers doubled available time for developing features by reducing rework and testing time to <10%

Improved quality and time-to-market, while eliminating work that adds no value at Intuit

“Engineers love it... Once they adopt it they can’t imagine going back”

Source: Jim Sartain, Director, Intuit
Adobe

Teams spent four times less rework than typical Adobe software projects

Teams removed 90% of total bugs prior to systems test

Teams plan, track, measure, and improve

Measurement has encouraged other improvements
  Improved design practices
  Defect analysis to determine common causes and implement improvement methods
  Better estimation methods

Source: The Evolution of Quality at Adobe, SEPG North America, March 20
AIS Federal IT Project Results - 1

Defects Removed by Phase

- Design
- Design Review
- Design Inspection
- Code
- Code Review
- Code Inspection
- Component Test
- Integration Test
- System Test
- User Acceptance Test

Plan vs. Actual

© AIS 2009
What does “FUN ON THE JOB” Mean to you?
Contact Information

Girish Seshagiri
Advanced Information Services Inc.
(703) 426 2790
Email: girish.seshagiri@advinfo.net
Website: www.advinfo.net