Leadership of Software Quality Assurance

Mark Bentsen,
CTAL, CSTE, PMP, ASQ CMQ/OE
QA Manager
ARGO At A Glance

- Founded in 1980
- Provides Mission Critical Software and Services
- Financial Services Best Practice Competency
  - Lending
  - Sales & Service
  - Teller Payments
  - Fraud
- Healthcare Best Practice Competency
  - Patient Credit
  - Enterprise Master Patient Identity
  - Patient Transition Care
- Financial Strength
  - Revenue: $47 Million
  - Balance Sheet: $192 Million Assets and No Debt
  - 31% of Revenue invested in R&D over the past five years

Experienced Management Team

**Executive**
- Max Martin, Chairman & CEO
- David Engebos, President & COO
- Todd Robertson, SVP Business Development

**Corporate Administration**
- Melissa Kirkham, VP Accounting & Finance
- Brian Jobe, VP General Counsel
- Jolene Hersch, Director HR

**Product Management**
- Irene Shippee, VP
- Craig York
- Calin Sandru, VP
- Randy Wynn

**Technology**
- Ted Martin, VP
- Cyrus Bavarian
- Jerry Bowman

**Implementation & Support**
- Brent Tompkins
- Don Perry
- Daniel Baez

Production Footprint
- 32,500 Retail Offices / 6,000 Contact Center Seats
- 301,500 Workstations
- 100,408,800 Daily Transactions

EDMS Monitoring
- 22,613 Financial Centers in 40 States
- 103,230 Workstations and Servers
- 69,338,000 Daily Transactions

*Based on ARGO Audited Financial Statements, FYE June 30, 2015*
What We Do

With research and development investments, we continuously improve our products and find inventive ways to use technology to achieve efficient operations.
ARGO has over 100 additional customer installations through our resale partners.
Mission Critical Support (EDMS) – the Results

- Predict: 62%
- Respond: 81%
- Resolution & Recovery: < 1 Hour (80%)
- Audit Checklist: Prevention (81%)
- ARGO Detector/First Responder: (62%)

© 2016 ARGO Data Resource Corporation. All rights reserved.
EDMS Monitoring

August 2015 - October 2015 EDMS Production Monitoring Statistics

- **7,600,000,000** Transactions Monitored
- **1,466** Transactions per Second - **3,760** @ peak
- Average Response Time = **.19 - .24** Seconds @ peak

EDMS Predicted:

- **28** Severity 1 Issues (Outage)
- **18** Severity 2 Issues (Degradation)
- Potential Outage Hours Prevented = **99,020** Hours

ARGO - First Responder to **86%** Events
## ARGO Technology Stack – Development View

<table>
<thead>
<tr>
<th></th>
<th>Sales &amp; Service Retail Lending</th>
<th>Teller/Payments</th>
<th>Commercial Lending</th>
<th>Fraud</th>
<th>HealthCare</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Presentation Tier</strong></td>
<td>HTML/JS</td>
<td>HTML/JS</td>
<td>HTML5 Semantic Web</td>
<td>Java</td>
<td>Angular JS</td>
</tr>
<tr>
<td></td>
<td>HTML5 MVC5 Angular JS</td>
<td></td>
<td></td>
<td>JSP</td>
<td>JQ Widgets</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.Net</td>
<td>Groovy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Java,</td>
<td>Grails</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C++</td>
<td>Hibernate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C/C++</td>
</tr>
<tr>
<td><strong>Data Tier</strong></td>
<td>SQL Server (ODBC)</td>
<td>SQL Server (ODBC)</td>
<td>SQL Server (ADO.Net)</td>
<td>SQL Server (JDBC/ODBC)</td>
<td>SQL Server (Oracle Postgre SQL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ARGO Decision Engine</strong></td>
<td>C/C++, C#, .Net, Lua, COM+ and SOAP Interoperability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Document Services</strong></td>
<td>C#, WPF .Net, SOAP Interoperability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Custom Integration / Interoperability</strong></td>
<td>SOAP, REST, HTTPS, TCP Socket, MQSeries, SNA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Presenter
Mark Bentsen

- Quality Assurance Manager, ARGO Data
  CTAL-Full, CSTE, PMP, ASQ CMQ/OE

- Leading cross-enterprise collaboration in the DFW area among Software Quality practitioners for the past seven years. Gathering corporately on a quarterly basis and on field trips to one another’s work places, the QA Trailblazers are pushing each other to expand the capabilities of the modern software testing organization.

- QA Manager of ARGO Data, a software development company providing mission-critical and analytical solutions for financial services and healthcare.

- Mark & his wife Melissa are the President Couple of Better Marriages Texas and have been active in Marriage Enrichment since they said “I do” in 2001. Prior to working in technology, he worked in YWAM & Mercy Ships in Switzerland and Namibia.

- Contact: Mark.Bentsen@argodata.com
What is Testing?

Testing is NOT QA

The Coffee Analogy

The problem:
You have coffee grounds in the coffee.

Solution:
- Tweeze the grounds out
- Use a better filter
- Learn how to use your tool
What do we do?

- Reduce Risk & Eliminate Waste

- Effective **software testing** teams:
  - Build confidence
  - Reduce “Risk & Surprises”
  - Detect defects early
  - Provide valuable information
What do we do?

- Reduce Risk & Eliminate Waste
- Effective Quality Assurance teams:
  - Identify risks
  - Prevent defects
  - Focus on continuous improvement of SDLC quality
  - Guard the company brand
What is Testing?

The Types of Testing (Green Circles)

Figure copyright and courtesy of Rex Black.
Advanced Software Testing Vol.2

© 2016 ARGO Data Resource Corporation. All rights reserved.
The two essentials of leading technical professionals
Priority management and scaling your effectiveness of project delivery
Credibility of the testing Organization

Credibility

Building and **keeping credibility**
- Credibility is based on trust built over time
- It can be lost in a moment
- Credibility is not perfection
- Be quick to admit mistakes and slow to make assumptions

**Damage Control** – **Rebuilding Lost Credibility**

Be honest and open
- Allow time to recover trust
- Keep relationships and lines of communication open
- Be able to explain your position. Don’t argue.
- Document your findings carefully
Test Leadership

I share my expectations for the team of testers.

Then they have the homework to share their expectations of me.
A Project = Who does What by When

\[ \text{Leci n'est pas une pipe.} \]
Five Whys

- Work backward from the problem to identify the root cause.
- Ask “Why does this happen?”
- For each answer ask why again.
- Continue until the reason is no longer related to the problem.
- Typically requires asking “Why” five times.
Non-technical Example

- I have a flat tire
- Because I have a nail in my tire
- Because I drove through a construction site on my way to work.
- Because it’s the only way to get to work.
- Root Cause: I have a flat tire because I drove through a construction site on my way to work and drove over a nail.
## Appendix I - Primary Contributing Cause

"Primary Contributing Cause", found in the **Quality Control Status** tab, captures the root cause for the defect. Additional supporting info is to be included in the defect’s **Comments**. Primary Contributing Cause is to be assigned during or before dev’s **Fixed** status.

<table>
<thead>
<tr>
<th>Root Cause – Primary Contributing Causes Defined</th>
<th>Quick Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Handling/Data Management</strong> – Data handled improperly, causing issues where data is not validated, defined, transformed, masked/encrypted consistent with ARGO published standards or specifications.</td>
<td><strong>Performance</strong> – Issue pertaining to memory leaks, data volume, architectural complexities, and ineffective processes, generally discovered during performance testing.</td>
</tr>
<tr>
<td><strong>Environment</strong> – Error produced by combination of hardware, configuration, or code version discrepancy. Includes compilation, build errors and failed code pushes where application runtime files not updated properly.</td>
<td><strong>Process Deficiency</strong> – Process in SDLC is incomplete, ambiguous or too tolerant of errors resulting in issues that degrade quality of deliverables, communication and permits defects to manifest in application.</td>
</tr>
<tr>
<td><strong>Exception Handling</strong> – Architectural, global or functional exception handling contaminated. Missed exceptions causing issues in otherwise issue-free logic. Unusual situations not handled non-destructively creating cascading issues.</td>
<td><strong>Serialization, Timing, Sequence</strong> – Issues exposed or created when dependencies between functions are not identified prior to subsequent development.</td>
</tr>
<tr>
<td><strong>Insufficient Skill / Knowledge</strong> – Issue that originated from programmer’s lack of skill or knowledge on line of business, application, or development methodology resulting in failure to effectively complete the task.</td>
<td><strong>Requirements Unclear / Inadequate</strong> – Gaps in functional requirement specifications and inadequate design definitions cascading into additional issues further in the SDLC.</td>
</tr>
<tr>
<td><strong>Logic / Decisioning Errors</strong> – Business logic not correctly interpreted programmatically. Application therefore does not follow decisions, policies, or explicit intent in specified requirements.</td>
<td><strong>Side Effect / Downstream Impact</strong> – Issue caused inadvertently while making changes either in development of another function or while addressing another issue.</td>
</tr>
<tr>
<td><strong>Failed to Conform to Specification</strong> – Issue transpired from developer misinterpreting accurately specified requirement, programming per own understanding, Application functions without error, but not as designed.</td>
<td><strong>Programming Error</strong> – Error that originated during development which caused specified requirement to not function as designed.</td>
</tr>
<tr>
<td><strong>Failed to Conform to Standards</strong> – Failed to conform to published standards for UI or other physical attribute behavior.</td>
<td></td>
</tr>
</tbody>
</table>
Appendix I – Decision Flow to Assign Primary Contributing Cause

Root Cause – Primary Contributing Cause Decision Flow
Sequential Development V-Model

Testing Throughout the Software Life Cycle

Testing in the lifecycle

Operational or business need
- Verify operational or business need

Define requirements
- Verify requirements

Design system
- Verify design

Build system
- Verify construction

Unit Test
- Validate construction

Integration test
- Validate Design

System test
- Validate requirements

Acceptance test
- Validate operational or business need

Test Execution (Static)

Test Execution (Dynamic)
Scrum Overview

http://www.mountaingoatsoftware.com/scrum_figures
Agile Testing V Model

Test Planning and Preparation

- Prepare Testing Strategy
- Hardening Sprint Test Plans & Cases

Test Execution

- End-to-End Regression Test, Performance Test (Business flows), UAT
- System Integration Test, Performance Test (System flows), UVT
- Unit Test, Component Integration Test, Service Test, Functional System Test
- Smoke Test

Development Life Cycle Activity

- User Stories
- SOA Framework
- Elaborated User Stories
- Feature Sprint Test Plans & Cases
- Construction (Coding)
- Release Plan and Product Backlog

Validation (Dynamic Testing)

Legend

- Development Life Cycle Activity
- Output
- Testing Activity
- Validation activity
Defect Flow Client Implementations

ARGO Defect Lifecycle

Possible Status Changes

1. Customer Test is an ARGØ status. At that point the client defect system is the system of record.
2. The "Reopen" and "Rejected" states are equivalent to "Research", and follow the same path.
3. The ARGØ Product Manager will assign WAD issue back to the client QA Lead with comments.
4. ADE & Fast Followers issues retested when ISV testing is complete or when fast followers are implemented. Status is "Pending".

© 2016 ARGO Data Resource Corporation. All rights reserved.
# Appendix III - Defect Governance & SLAs

**Lifecycle of Development & Testing – SLA’s & Defect Business Priority**

To identify issues in ALM that exceed defined service level agreement
To place additional focus on older issues that may not be valid due to product direction or implemented enhancements.

<table>
<thead>
<tr>
<th>Timeline Events</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actors</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST</td>
<td>QA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Client or ARGO QA encounter a suspected defect.</td>
<td>Client or Product sets defect business priority.</td>
</tr>
<tr>
<td>Defect written up in defect tracking module.</td>
<td>1 – Immediate</td>
</tr>
<tr>
<td>2 – High</td>
<td></td>
</tr>
<tr>
<td>3 – Medium</td>
<td></td>
</tr>
<tr>
<td>4 – Low</td>
<td>Primary Contributing Root Cause is discovered during research.</td>
</tr>
<tr>
<td>Determine solution.</td>
<td></td>
</tr>
<tr>
<td>Develop code.</td>
<td></td>
</tr>
<tr>
<td>Unit Test Code Review</td>
<td>Developed code is ready for release.</td>
</tr>
<tr>
<td>Dev. assigns Primary Contributing Root Cause.</td>
<td>Developed code packaged for delivery to QA.</td>
</tr>
<tr>
<td>QA performs confirmation testing for defect resolution.</td>
<td></td>
</tr>
<tr>
<td>If defect still exists, it is returned to Dev.</td>
<td>QA closes resolved defect.</td>
</tr>
<tr>
<td>Client confirms defect resolution.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Research</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SLA’s</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30 Days</td>
<td>(I &amp; H) 5 Days</td>
</tr>
<tr>
<td>(M &amp; L) 30 Days</td>
<td>(M &amp; L) 30 Days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contingencies</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(SLA: 5 Days)</td>
<td>Defect Exist</td>
</tr>
<tr>
<td>(Status: Rejected)</td>
<td>Defect Exist</td>
</tr>
</tbody>
</table>
Requirements Quality

The Key to Quality
Static Techniques and the Test Process

Most defects are introduced in the requirements

Typically, the defects introduced in the requirements remain undetected until the test execution phase, or worse still, until the developed system is delivered to the customer, because the original undetected defect also drives incorrect design, code development, and test case development.

The amount of effort (and the corresponding cost) that it takes to fix defects whose origin can be traced to the requirements is even higher at 82%.
Static Techniques and the Test Process

Relationship between Requirements, design, and code

An error in requirements must be corrected not only in the requirements themselves, but also in the design, the code, and the test cases. In other words, the rework effort can almost equal the initial design, development and testing effort.
The typical defect discovery rate on projects that rely exclusively on code-level testing to validate application quality, and do not perform rigorous reviews of upstream deliverables is 85%.
Cost of Defects

Myths & Realities
Cost of Defects at Different Stages of the SDLC

Source: Capers Jones, *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley, 2000
Phase That a Defect Is Corrected

McConnell, Delivering Software Project Success: 10 Myths of Rapid Development, 2001

BST Defect Cost Analysis
Pressman Cost Model

Requirements: 1X
Code: 10X
Early Test: 15-40X
Late Test: 30-70X
Production: 40-1000X

IBM Cost Model

Relative Costs to Fix Software Defects

IBMS Defect Cost Analysis

IBM Systems Sciences Institute

BST Defect Cost Analysis
## Components of Cost of Software Quality (CoSQ)

<table>
<thead>
<tr>
<th>Cost of Prevention</th>
<th>Cost of Appraisal</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Solid requirements</td>
<td>• Work product reviews</td>
</tr>
<tr>
<td>• Management of quality &amp; process improvement</td>
<td>• Code reviews</td>
</tr>
<tr>
<td>• Training</td>
<td>• Testing</td>
</tr>
<tr>
<td>• Automation</td>
<td>• Audit and compliance activities</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost of Internal Failure</th>
<th>Cost of External Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Analysis</td>
<td>• Service Failures</td>
</tr>
<tr>
<td>• Defect repair</td>
<td>• Reputation impact</td>
</tr>
<tr>
<td>• Crisis management –</td>
<td>• Crisis management - Ops</td>
</tr>
<tr>
<td>• Project Time/Costs</td>
<td>• App Support</td>
</tr>
<tr>
<td>• Re-testing</td>
<td>• Customer Service calls</td>
</tr>
<tr>
<td>• Opportunity costs related to missing launch dates</td>
<td>• Defect remediation</td>
</tr>
<tr>
<td></td>
<td>• Regulatory non-compliance</td>
</tr>
</tbody>
</table>
Defect Analysis - Cost per Phase

On average, 10 people touch each defect

* Estimate
# Defect Cost Analysis Results

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low/Medium</td>
</tr>
<tr>
<td>Total Average Defect Cost</td>
<td>$293</td>
</tr>
<tr>
<td>Requirements</td>
<td>$50</td>
</tr>
<tr>
<td>L1 Defect Cost</td>
<td>$120</td>
</tr>
<tr>
<td>L2 Defect Cost</td>
<td>$340</td>
</tr>
<tr>
<td>L3 Defect Cost</td>
<td>$420</td>
</tr>
<tr>
<td>Production*</td>
<td>$5,000</td>
</tr>
</tbody>
</table>

*Production used factor of 100x as an estimate, Production includes defect correction, customer impact, & lost revenue. Utilizing industry standard- low end weighting

All values averages and rounded to nearest whole number
January Analysis- Detecting Defects Earlier

- 50% defect shift saves $2.14MM per Corp Release
  - Finding 50% of each phases defects in earlier phase
Test Progress Monitoring and Control

The Value of Testing – Capers Jones

Another poor measurement practice that has concealed the economic value of software quality is the usage of the cost-per-defect metric. It has become an urban legend that “it costs 100 times as much to fix a bug after delivery as during development.” Unfortunately, the cost-per-defect metric actually penalizes quality and achieves its lowest values for the buggiest software. As quality improves, cost per defect rises until a level of zero defects is reached, where the cost-per-defect metric cannot be used at all.

The real economic value of high quality is only partially related to defect repair costs. It is true that high quality leads to fewer defects and therefore to lower defect repair costs. But its major economic benefits are due to the fact that high quality

- Reduces the odds of large-system cancellations
- Reduces the odds of litigation for outsourced projects
- Shortens development schedules
- Lowers development costs
- Lowers maintenance costs
- Reduces warranty costs
- Increases customer satisfaction
The value proposition of unit/component testing

How to Get development Peers ‘On-Board’ with Quality Practices in Development Workflows.
Structure-based or White-box Techniques

Describe the concept and value of code coverage

Structural Test Coverage Levels
Equivalence & Boundary | Positive & Negative

∞ -1 0 1 2 3 4 5 6 7 8 9 10 11 ∞

Positive

Negative

Boundary
The Riskiest of the Risks

“It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so.”

Mark Twain
There are five fundamental test activities and respective tasks from planning to closure.
Your Byproducts

Increase the Effectiveness of Your Test Coverage & Improve Code Quality
NO!!

Don’t Agree to the Impossible.
Communication Heuristics

- Misconception is that status and metrics only go out in email.
- If you depend on email, you have one “where” and one “how” in your communication tool belt. There are a lot of other tools available to the wise test manager.
Test Management Trifecta

- What have you completed?
- What did you learn?
- What remains?
BLUF

- Bottom Line Up Front
- Follow with a “Headline”
- Impact to the triple constraint?
“When documents are mostly to enable handoffs, they are evil. When they capture a record of a conversation that is best not forgotten, they are valuable.”

- Tom Poppendieck
## Unit Test - Maturity Model

<table>
<thead>
<tr>
<th>CMM</th>
<th>Unit Test Level</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0 - Unaware</td>
<td>Unaware of unit testing concepts or missing fundamental skills to develop unit test.</td>
<td></td>
</tr>
<tr>
<td>Level 1 - Ignored</td>
<td>A belief that not enough time is available for unit testing or that it would not bring benefit to the specific work at hand.</td>
<td></td>
</tr>
<tr>
<td>Level 2 - Experimental</td>
<td>Experimentation of basic unit test concepts, typically positive scenarios. Missing strategy as to coverage areas. Typically used by creator of test and not others within the organization. Likely not maintained for reuse...</td>
<td></td>
</tr>
<tr>
<td>Level 3 - Intentional</td>
<td>Intentional effort to build some unit test in places throughout the development lifecycle. May not represent test scenarios outside positive (happy path) testing.</td>
<td></td>
</tr>
<tr>
<td>Level 4 - Positive/Negative Test</td>
<td>Intentional effort to build positive and negative unit test throughout the development lifecycle. Understanding of testing principals beyond positive (Happy Path) testing techniques.</td>
<td></td>
</tr>
<tr>
<td>Level 5 - Positive/Triangulation Test</td>
<td>Specific test with different input and expected results than the positive test to ensure no hard coded return results.</td>
<td></td>
</tr>
<tr>
<td>Level 6 - Positive/Negative/Boundary Test</td>
<td>Intentional effort to build effective unit test leveraging appropriate testing principals such as Positive, Negative and Boundary testing. Effective communication channels in place between development and QA.</td>
<td></td>
</tr>
<tr>
<td>Level 7 - Mocks and Stubs</td>
<td>Mocks and Stubs in place to replicate dependent functionality.</td>
<td></td>
</tr>
<tr>
<td>Level 8 - Designed for Testability</td>
<td>Code that is easier to test due to development design. Clear delineation and simplicity in design.</td>
<td></td>
</tr>
<tr>
<td>Level 9 - Test Driven Development</td>
<td>Begin development process by building unit test which evolve with primary code development. Designed for testability. Red, Green, Refactor. Never write a line of code that doesn't have a failing test.</td>
<td></td>
</tr>
<tr>
<td>Level 10 - Code Coverage</td>
<td>Intentional effort to build unit test to measurably cover functionality, logic and lines of code across the development.</td>
<td></td>
</tr>
<tr>
<td>Level 11 - Unit Test in the Build</td>
<td>Automated unit testing during the build process (CI). All Unit Test must pass in order to consider the build successful.</td>
<td></td>
</tr>
<tr>
<td>Level 12 - Code Coverage Awareness</td>
<td>Awareness of Unit Test code coverage across an organizations landscape ensuring consistency in testing practices. High level dashboards showing metrics down to individual projects regarding code coverage and last execution times.</td>
<td></td>
</tr>
<tr>
<td>Level 13 - Automated Builds and Tasks</td>
<td>Fully automated build and reporting process. Bringing awareness to the collective and individual health of the SDLC process.</td>
<td></td>
</tr>
</tbody>
</table>