



QAI /QAAM 2011 Conference
"Proven Practices For Managing
and Testing IT Projects"



Design for Testability

**Co-Presented by Mr. Bill Rinko-Gay
and Dr. Constantin Stanca**

9/28/2011

Format

- This presentation is a journey
 - When Bill and Cristi started working together neither QA nor Development were achieving their goals
 - Bill and Cristi had to overcome some typical issues between the QA and Development teams
 - When the basics were resolved, Bill left the project, but Cristi improved the Development responsibility to create Quality software
 - The first half of this presentation will talk about some of the issues Bill and Cristi worked through, with opportunities to explore some of your solutions
 - In the second half, Cristi will focus on what it means to “design for testability”

Game

Project: Design a loft (BR, LR, BTH, KITCHEN)

- Assemble teams
 - 2 minutes
- Each team defines:
 - 5 functional requirements
 - 2 non-functional requirement
 - 5 minutes
- Each team designs several test cases to test a floor plan design vs. requirements
15 minutes
- Analyze results of the task. Was it successful or not? Were we fell short?
15 minutes
- Would you change design after knowing how to test?
 - 10 minutes

Session Abstract

QA Experts know that it's better to build quality in than to test defects out. But Developers have been slow to understand their role.

Bill Rinko-Gay, QA Manager, and Dr. Cristi Stanca, Development Manager, worked together for 14 months developing an understanding of the role the Development group plays in creating good software.

In "Designing for Testability" Bill and Cristi will share what they learned, and how Quality Analysts can define their needs in terms Developers can understand.

Bill and Cristi will share on topics like:

- An actual case study that reduced the time between releases from 9 months to 6 weeks with higher quality Developer test vs. Unit test
Creating a feedback loop to help developers improve
- Why testable software is high quality software Developer metrics to show a build is ready for test

Attend this session to take your relationship with your Development team, and your Software Quality, to the next level.

First issue:

Getting acceptable builds

- This was a discussion around the quality that should be there before a build is released to test
- Developers did not have a contract for specific build quality
- QA was concerned when new features worked badly and regression defects were introduced
- Lead to a distrust of the Development team

First resolution:

Getting acceptable builds

- Arrange to share QA System Tests with developers
- Discuss what is tested by QA in advance so testers understand what is acceptable
- Developers moved from Rational Robot to QTP / QC so we could share tests more directly
 - Originally, each group chose its tools independently

Second issue:

Writing useful defects

- Developers were having to do too much back-and-forth to fix defects
- QA was writing defects that weren't defects
- Development would not always prioritize the defects correctly because of incomplete information
- Defects were being fixed late (at the last minute) because of incomplete information
- Lead to a distrust of the QA organization

Second resolution:

Writing useful defects

- The QA Manager provided training to the QA Analysts to write better defects – and added a review step
- The QA team added required fields to help ensure specific information is included (i.e. found in build number, regression - build or release, last time tested, blocking)
- A Release Manager reviewed all defects with QA to make sure they were properly understood
- Developers added Root Cause Analysis, Test Impact Analysis, and created a Unit Test to reproduce the issue, and additional unit tests to increase coverage

Third issue: QAEs have insufficient understanding

- QAEs were logging too many defects that weren't defects
- Developers were taking too much time bringing QAEs up to speed on the new features
- QAEs were falling back on the excuse of “the tests were reviewed” instead of taking responsibility for good testing

Third resolution: QAEs have insufficient understanding

- Start engaging QAEs earlier in the feature definition
- Have QAEs review scenarios instead of test steps to ensure sufficient understanding
 - Take responsibility for good steps
 - Easier for reviewers to engage at the scenario level
- Have defects reviewed before assigning them to developers
 - QA internal review – completeness and test severity
 - Dev internal review – quality of defect documentation
 - Group review (BA, Dev, QA) - priority

Fourth Issue:

Incomplete analysis of changes

- Developers were introducing too many regression defects
- Developers were focusing on code effort but not on test effort or impact
- Developers were making changes without awareness of the schedule (minimizing risk late in the cycle)

Fourth resolution:

Incomplete analysis of changes

- Development team implemented a thorough root cause analysis
- Development team worked with Test team to identify the full cost of the change
- Development team took the schedule risk into account
 - Sometimes put a band-aid in place in the current release with the fix to follow in a later release

Fifth issue:

Understanding what is testable

- Developers wanted to release partial builds so Testers could provide early feedback
- Testers struggled to understand what was ready for test in a build – entered defects against functions that were not completed
- Test and QA had a difficult time describing what it meant for a feature to be “testable”

Fifth resolution:

Understanding what is testable

- Developers identified features as ready for QA feedback even though that was only partial functionality
- Testers select test cases applicable to testable features
- Features were isolated through sophisticated configuration management (multiple dev branches, merges, build from release branch) to minimize non-testable features

Are we testing the right things

- Aspects (requirements, design, standards, code etc)
- Timing (is it ready?)
- Priority (business or technical)



Break

Flaws in Development Process

- What is the main cause of most of the bugs in the software?
 - Misunderstanding requirements
 - Improper understanding of the deployment environment
 - Lack of adequate testing
 - Lack of regression testing
 - Bad execution of design
 - Coding errors

Flaws in Development Process (Continued)

- What is the main cause of schedule delays?
 - Inadequate requirements
 - Requirements creep
 - Change in architecture
 - Too busy fixing bugs
- How does your process react to changes in requirements?
- In what phase (design, coding, integration, etc.) do you have the most problems?

Quality Snapshot

What were developers seeing over the fence?

- Poor quality test cases:
 - Misunderstanding of requirements
 - Test cases had no business importance defined
 - Some automation but poorly scripted (buggy and slow)
 - No automation framework, non-existent reusability (just copy and paste)

Quality Snapshot (Continued 2)

What development manager saw in his yard?

- Very little unit test and low value
- No unit test frameworks or test harnesses
- No guidelines for unit test
- Very little culture for developer test and good quality unit test
- Code was not easy to test because of a low testable design
- Developers excuses for not unit testing

Developer's Excuses for Not Testing

- It takes too much time to write the tests
- It takes too long to run the tests
- It is not my code, I just made a minor change
- I have so many features to develop that I will get to unit test some other time
- I feel guilty about putting testers and QA staff out of work 😊
- *Can anyone add others from experience working with developers?*

Testability

- **Observability** is the ability to observe an output of a function, or how easy it is to tell what the program is doing
- **Controllability** is the ability to apply any and all desired inputs to a function, or how easy it is to drive the program where we want to go
- To test a function completely, all combinations of inputs must be applied and the output corresponding to each input must be observed

Design for Testability

- Most of developers believe that to create good software requires a good design
 - You cannot come-up with a good design just by thinking about it
 - You can go through RA, create data models and other great theoretical constructs, but the minute you start coding, a lot of that stuff gets thrown out of the window and forgotten
 - It is impossible to make all decisions before writing the software
 - A design that is difficult to test will increase cost of achieving quality verification

Design for Testability (Continued 1)

- A design for testability generates code that is easy to test, assuring high coverage and low cost
- Continuous refactoring for testability
- Improved design:
 - Better encapsulation and modularity
 - Simpler class relationship
 - Reduced design complexity

Design for Testability (Continued 2)

- Design optimizes across multiple dimensions
 - Accuracy
 - Flexibility
 - Speed
 - Usability
 - TESTABILITY
 - Software can be tested easily
 - Encapsulation
 - Design to contract

Design for Testability (Continued 3)

- Refers to design styles that reduces test generation complexity
- Motivation: Test generation complexity increases exponentially with the size of the function
- Prepare for testing before we write any code:
 - testing as an after-the-fact, ad hoc, exercise is often limited by earlier design choices
 - Write automated test cases first
 - Then write the code to satisfy tests
 - This approach forces observability and controllability

Test Best Practices

- Keep it Simple
- Make it easy to run
- Minimize test dependencies
- Keep a test self-contained
- Properly document the test

GUI Testing

- GUI tests need to be structured to test the functionality of the display, not the validity of the data being displayed
- Mostly black-box testing
- Configuration-driven testing

Dev QA Improvement Approach

- Hired expert
 - Defined process
 - Defined guidelines
 - Training sessions
 - Analyzed code for best value coverage with unit test
- Changed code for testability
 - Testability injection points at architectural level
 - Design for testability
 - Refactor for testability

Dev QA Improvement Approach (Continued 1)

- Implemented test frameworks and harnesses
- Implemented dynamic test data framework
- Automated code coverage analysis
- Automated code practices
- Automated unit test generation
- Developed a logging utility to extract input/results of QTP tests and convert to unit test harness execution:
 - All functional test cases were executed with each build (several builds/day), 100x times faster, and not only every two weeks
- Released only 100% unit test passing build to System Test
 - No flexibility for bad quality builds even that meant some delays initially and bringing on-board management and developers

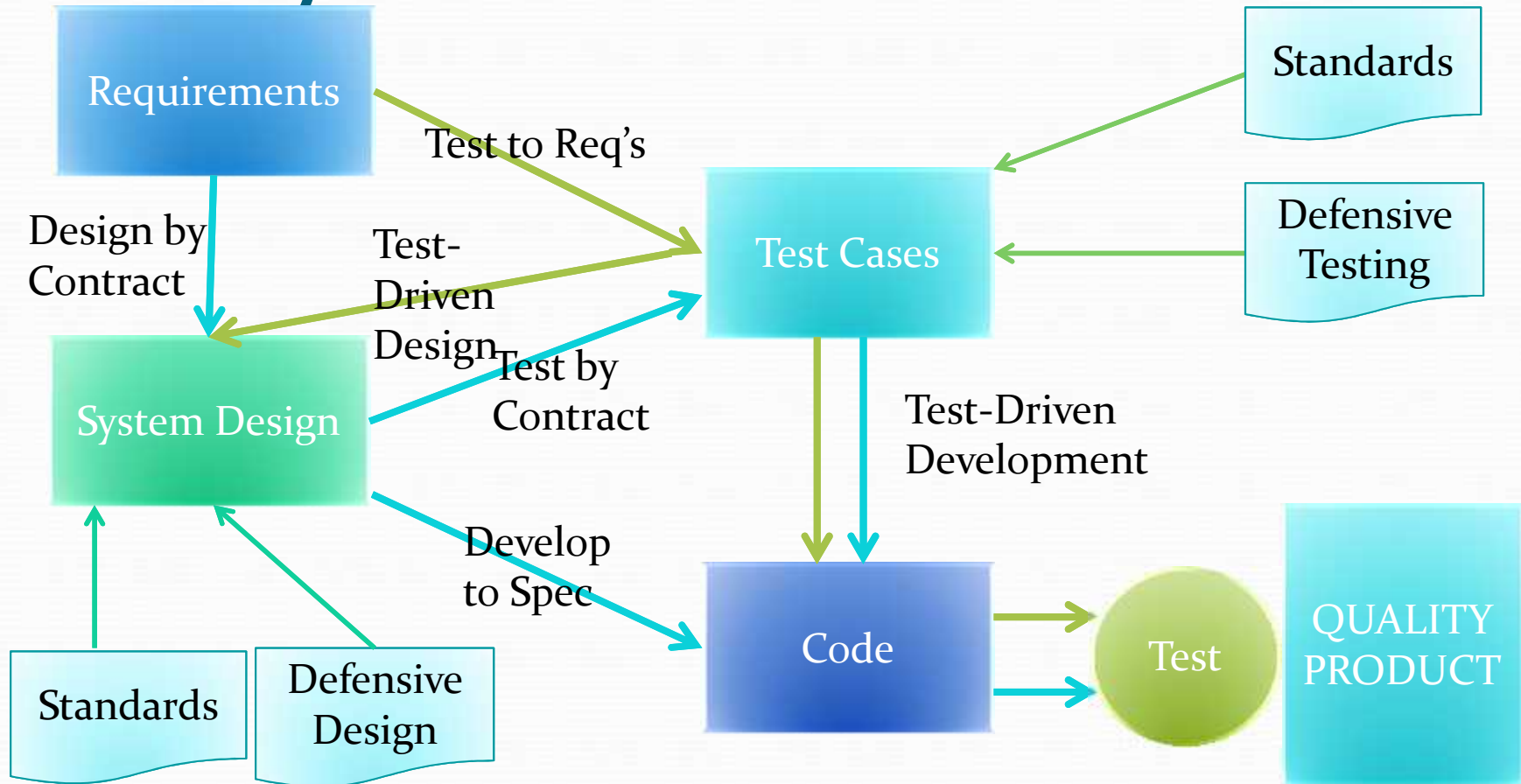
Dev QA Improvement Approach (Continued 2)

- Converted all long running test cases to fast running cases
- GUI testing using Selenium-based test harness
 - Layout validation
 - Configuration-driven
- Reliable test data management
- Stability Tests
 - Identified common steps
 - Weighted each step
 - Critical path for highest coverage and lowest execution time

Dev QA Improvement Approach (Continued 3)

- Complementary Quality
 - Unit test
 - Coding practices
 - Developer integration test
 - Developer test harnesses
 - Reduced redundancy testing
 - Test more
 - Test sooner
 - Test in lesser time
 - Test what is important

Quality Product Value Chain



What Should You Ask Developers?

- 100% passing unit test builds with a certificate of quality from development point of view (unit tests, integration tests, smoke test, coding practices, coverage, etc)
- Ask to open the box and explain how and which unit tests are mapped to what feature
- Unit test strategy included in system design
- Test harnesses to test key functions
- GUI testing

What else would you like developers to do to improve QA?

Questions

- Dr. Cristi Stanca
 - constantin@stancas.com
- Bill Rinko-Gay
 - billrg@earthlink.net