Preparing the Next Testers: An Undergraduate Course in Quality Assurance

PACIFIC NW SOFTWARE QUALITY CONFERENCE
15 OCTOBER 2013
PETER A. TUCKER, WHITWORTH UNIVERSITY
PTUCKER@WHITWORTH.EDU

Talk overview
Course overview
Tips, Tricks, and Foundational Techniques
Five Views of Software Quality
Conclusion & Questions
Course Overview

- Used heavily for fundamental QA techniques

Textbook: Kaner, *Lessons Learned in Software Testing*
- Students looked forward to these, we’ll even end today with two

10-week testing project, software from SourceForge
- Teams of 2-3
- Bug reports, test plan, automated regression tests

Target student: sophomore CS major

First half: general tips, techniques, and tools
Second half: Quality views per Garvin

Students need guidance on how to write tests: tools and tips in QA

Each test case should execute only one thing, and be self-contained
- If you’re testing that `vector::push_back` works for a zero-length array, your test should create a vector, add a single item, check that the item is there and that there’s only one item, then exit

Test cases should clean up after themselves
- If your test creates a temporary file, delete it

Be consistent in naming conventions

Look for code shared across tests
- Great code for a library
- Test case development can lead to copy/paste more easily than product code development – don’t let it
Visual Studio.NET
Unit testing and Coded UI Testing

```csharp
[TestMethod]
public void TestValidAddToEmptyList()
{
    List<string> items = new List<string>();
    string st = GenRandomString();
    items.Add(st);
    Assert.AreEqual(1, items.Count);
    Assert.AreEqual(st, items[0]);
}
```

```csharp
[CodedUITest]
public class CodedUITest1
{
    ApplicationUnderTest testapp;

    // Use TestInitialize to run code before running each test
    [TestInitialize()]
    public void MyTestInitialize()
    {
        testapp = ApplicationUnderTest.Launch(@"C:\windows\system32\calc.exe");
    }
}
```

Students should know how to write a bug: Anatomy of a bug report

| Field          | Description
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Short, descriptive text summarizing the bug. IMPORTANT: This is read by many people, so must describe the bug carefully</td>
</tr>
<tr>
<td>Description</td>
<td>Notes on the bug not obvious from the title. Customer impact, discussions with developer/program manager, ...</td>
</tr>
<tr>
<td>Status</td>
<td>Unconfirmed, New, Assigned, Resolved, Closed (life cycle)</td>
</tr>
<tr>
<td>Version found</td>
<td>What version was the bug found in?</td>
</tr>
<tr>
<td>Reproduction steps</td>
<td>Specific series of steps to reproduce the bug. Take extra care here</td>
</tr>
<tr>
<td>Assigned</td>
<td>Who's currently working on this (dev, test, pm)</td>
</tr>
<tr>
<td>Severity</td>
<td>1 – crash, 4 – trivial</td>
</tr>
<tr>
<td>Priority</td>
<td>1 – blocking testing, 4 – few people care</td>
</tr>
<tr>
<td>Environment</td>
<td>Notes on the environment the test was run in (e.g. OS, browser, ...)</td>
</tr>
<tr>
<td>Resolution</td>
<td>Fixed, Dup, Not Repro, ... (lifecycle)</td>
</tr>
<tr>
<td>Regression test case</td>
<td>If closed, make sure there's a test that covers this</td>
</tr>
</tbody>
</table>
Foundational Skills – Running example

From Glenford Myers

```c
enum TRIANGLE { Scalene, Isosceles, Equilateral, Right, NotATriangle };

TRIANGLE IsTriangle(int a, int b, int c)

Note: a, b, and c must be in [1, 200]
```

Students should know what tests to write: Foundational Skill – Boundary Testing

Focus testing at the object’s boundaries
Boundary Tests for IsTriangle

```c
enum TRIANGLE { Scalene, Isosceles, Equilateral, Right, NotATriangle };

TRIANGLE IsTriangle(int a, int b, int c)
◦ Note: a, b, and c must be in [1, 200]
```

Students should know what tests to write: Equivalence Class Testing

Find classes of tests, where each class in the test would be “equivalent”
◦ That is, running multiple tests from the same class could be redundant

Weak vs. Strong
◦ Weak runs tests across one dimension of equivalence
◦ Strong runs tests across all dimensions

Normal vs. Robust
◦ Normal runs tests with legal values
◦ Robust includes illegal values
Weak vs Strong (Normal)

\[ f(x_1, x_2), \text{ where } a \leq x_1 \leq d \text{ and } e \leq x_2 \leq g \]

Let’s define partitions as follows:

for \( x_1 \), \([a, b), [b, c), [c, d]\]

for \( x_2 \), \([e, f), [f, g]\]

Weak vs Strong (Robust)

\[ f(x_1, x_2), \text{ where } a \leq x_1 \leq d \text{ and } e \leq x_2 \leq g \]

Let’s define partitions as follows:

for \( x_1 \), \([a, b), [b, c), [c, d]\]

for \( x_2 \), \([e, f), [f, g]\]
Equivalence class testing for IsTriangle

```c
enum TRIANGLE { Scalene, Isosceles, Equilateral, Right, NotATriangle };

TRIANGLE IsTriangle(int a, int b, int c)
◦ Note: a, b, and c must be in [1,200]
```

The enum really gives me at least some of the classes I’ll want to try
◦ <5,5,5> is likely the same test as <10,10,10> → Equilateral class

Students should see different views of software quality

- **Manufacturing** – quality of the development process
- **Product** – quality features of the product
- **User** – user satisfaction with the product
- **Value** – tradeoff between quality and cost/price
- **Transcendental** – quality is obvious to those who use it

The second half of class is divided into discussions on each of these views
◦ Discuss meaning, importance, and how they play into the tasks of quality assurance
Manufacturing View:
How well is the product built?

What practices were used in building the software?
- ISO/IEC 25010:2011
- CMM

What tools were used?
- Code coverage, Profiling, Test automation, Source code control
- Roseberry

Test plans
Code and Test reviews
- Spinellis Code Reading

Static analysis tools
- Code Analyze in VS.NET, Lint, Resharper
- Ruberto

Initial Test Plan

<table>
<thead>
<tr>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software, including version and release numbers</td>
</tr>
<tr>
<td>Document Revision History</td>
</tr>
<tr>
<td>Table of Content</td>
</tr>
<tr>
<td>Purpose of the document</td>
</tr>
<tr>
<td>Intended audience</td>
</tr>
<tr>
<td>Software overview</td>
</tr>
<tr>
<td>Naming Conventions</td>
</tr>
<tr>
<td>Test Outline</td>
</tr>
<tr>
<td>Testing approaches used by feature, functionality, process, ..., as applicable</td>
</tr>
<tr>
<td>Boundary value analysis, equivalence classes, decision tables</td>
</tr>
<tr>
<td>Test automation</td>
</tr>
<tr>
<td>QA Tools to be used</td>
</tr>
</tbody>
</table>
Product View:
What are the quality features of the product?

Are there features of this product that make it unique?
Is the software more efficient, in terms of speed or memory?
Are there security features that make it a higher quality product?

common software metrics: bug counts, bug rates, code coverage results, and mean time to failure.

Practices
- Code coverage, performance testing, profiling
- Security testing, abuse scenarios

Exploratory testing
I fit exploratory testing in product view
- Discover the unique, important, or risky features of the product
- Better fit in User view (next)

Students were given tips and direction
- The tester controls the design of tests as they are being performed
- The tester uses information from executing tests to design new tests
- Tour bus testing (Kaner)
  - You have a plan: a start, a finish, and points along the way
  - You can stop at various places and explore
  - Key: Don’t get lost – that is, get back on the bus soon
  - Key: Don’t fall asleep – that is, be attentive and curious as the bus travels
As a class *mission*: test the Filters options in GIMP, including functionality, stress, and boundaries

Skills for exploratory testing are important for many QA tasks

*Test design*: ability to craft tests that explore the product. Skills: analyze a product, evaluate risk, use tools, think critically

*Careful observation*: watch for *anything* unusual or unexplainable

*Critical thinking*: review and explain your own logic in the path you take: why did you test in a particular direction?

*Diverse ideas*: experienced testers generate test ideas from their own experience. They have seen enough to have developed “standards”. (e.g. pass NULL to pointer parameters, always use 0, 1, -1, ...) Many other guidelines are helpful: [http://www.satisfice.com/tools/satisfice-tsm-4p.pdf](http://www.satisfice.com/tools/satisfice-tsm-4p.pdf)

*Rich resources*: deep inventory of tools, information sources, data, and friends
User View: How satisfied is the user with the product?

Five “-ilities”
- **Functionality** – we've addressed this as part of the product view
- **Reliability** – how well does the program perform in the “wild”
- **Usability** – how easy is it for users to use your product
- **Maintainability** – how hard is the program to maintain?
- **Portability** – does the program run on multiple kinds of systems

Practices
- Ad hoc testing, app contests, Bug bashes, Beta releases
- Stress testing
- Performance testing (perfmon)

Value View: How much does quality cost?

Customers may be willing to pay less for a product that isn’t tested as heavily
- It’s still considered quality
- Compare quality between a Mercedes and a Honda

Resolving tensions between quality, time, resources, and scope

The Economics of Software Quality

Practices
- Technical debt
Transcendental View:
“But even though quality cannot be defined, you know what Quality is.” (Pirsig)

Practices
◦ Good, experienced people
◦ Ad hoc testing (e.g. app contests), “dog food”
◦ Honest beta customers

Testing Quality In (Emory)

Final Test Plan

- Introduction
  - Title
  - Software, including version and release numbers
  - Document Revision History
  - Table of Content
  - Objective of Testing Effort
- Software product overview
  - Relevant related document list, such as requirements, design documents, other test plans, etc.
  - Relevant standards or legal requirements
- Overall software project organization and personnel/contact-info/responsibilities
  - Test organization and personnel/contact-info/responsibilities
  - Assumptions and dependencies
  - Testing priorities and focus
  - Scope and limitations of testing
- Test outline - a decomposition of the test approach by test type, feature, functionality, process, system, module, etc. as applicable
  - Equivalence classes, boundary value
  - Test environment
  - Code reviews and test reviews
  - Code coverage
  - Exploratory testing
  - Security testing techniques and tools
- Non-functional Testing
  - Reliability
  - Usability
  - Maintainability
  - Portability
- Test automation and tools - justification and overview
  - Test tools to be used, including versions, patches, etc.
  - Test script/test code maintenance processes and version control
  - Problem tracking and resolution - tools and processes
Evaluation, Results, Next Steps

Student feedback
◦ Useful class
◦ Helps to improve design and development, and new skills for testing
◦ Tools are hard to come by

Results
◦ Students are getting QA jobs, as well as dev and PM
◦ Companies notice our students’ resumes that have QA

Working on a textbook to match this class structure

Kaner Lesson #16

“Testing is applied epistemology"

Epistemology is the study of how you know what you know
◦ Empiricism, innatism, intuitionism, authoritarianism

Applied to software testing, epistemology asks:
◦ How do you know the software is good enough?
◦ How would you know if it wasn’t good enough?
◦ How do you know you’ve tested enough?
Kaner #17

“Studying epistemology helps you test better”

Epistemological topics that relate to testing:
- How to gather and assess evidence
- How to make valid inferences
- How to use different forms of logic
- What it means to have a justified belief
- Differences between formal and informal reasoning
- Common fallacies in informal reasoning
- Meaning and ambiguity in natural language
- How to make a good decision

Questions?

Peter A. Tucker, Whitworth University
ptucker@whitworth.edu
Lifecycle of a bug

http://www.bugzilla.org/docs/2.18/html/lifecycle.html