DEPARTMENT OF COMPUTER & INFORMATION SCIENCES

University of Strathclyde Science

Software Engineering

Lecture 7: Testing

Billy Wallace
w.wallace@strath.ac.uk

CS993

Lecture Outline

- What is testing?
- What kinds of testing are there?
- How do we know when we’ve finished with testing?
- Testing tools

- “Testing shows the presence, not the absence of bugs.”
    - Dijkstra

How many bugs are left?

- People talk about “seeding” but no-one does this.
- Instead you stop testing when you have a good feeling that things are robust.

Focus on Risk

- Risk = cost x likelihood
- So focus testing where the cost of failure is high so that you reduce the likelihood of failure and so reduce the risk.
- Example - age appropriate recommendations for television companies.

Ship it!

- The fact that some bugs are low impact means that we will ship with known bugs.
- Having a work-around reduces the cost to the customer.

What exactly is testing?

- Verification
- Validation
- (Qualification)

Validation

- Did we build the right thing?
- Show that we met requirements
  - User Acceptance Testing (UAT)
  - System testing(?)

Validation

- Start thinking about testing early.
- What are the tests we’ll use for UAT?
- Document this so that requirements can be tracked right through to testing.

Validation Example

- Building the wrong thing is a very common way for a project to fail.
- At a bank, I prototyped a system to recommend a product to promote, if a call to the credit card call centre went well.
- The system worked perfectly, except that it used the wrong rules to show whether a user was eligible for a product and so very few offers were made.
- Of the offers made, most were accepted by the users, so if we had trialled the system with the more lenient rules, the trial would have been a huge success.
- At the end of the day we didn’t produce the evidence needed to show that the product would be a success and so the full system was never commissioned.

Verification

- Did we build the thing right?
- Here’s where we worry about bugs

Levels of Testing

- Unit testing
- Integration testing
- System testing
  - Non-functional usually goes here

Some Terminology

- Black box testing
  - Think about TDD
- White box testing
  - Required for boundary value analysis

Some more terminology

- Alpha and beta testing
- Regression testing
- Performance testing - test performance on real systems, patterns, data, etc.
- Environments: dev, test, stage & live

Test Coverage

- Boundary value analysis lets us find values that will drive the code down different paths.
- Coverage tools will show how much of the code is covered by test cases.
- Exception handlers can be difficult to cover.

When bugs are reported

- It can be difficult to identify the cause.
- Ask for all the details, have a template for this.
- It can be useful to leave tests or some form of instrumentation in the software to allow the customer to be able to give you better information about the problem.

Fixing the bug

- Start by creating a test that reproduces the bug. Easier said than done sometimes.
- After fixing the problem, keep the test as a regression test.
- A good way to spend a long time debugging is to spend too little time unit testing.
- If you do testing at the end, expect to be working long, stressful hours testing while the customer is unhappy because you promised delivery last week!

Egoless Programming

- Everyone breaks things at some point. It’s important that this doesn’t break the team.
- Everyone is responsible for software quality, no single person is to blame.

Tools: JUnit

- Test Cases
- Test Suites
- Eclipse will give you a nice red/green report on the test suites.
- Make sure unit tests are quick. Ideally tests don’t touch disk, network or anything else that’s slow.

Tools: Mock Classes

- JMock
- Mockito
- Good testing will drive good design. We can’t create mock classes when we don’t have interfaces to mock.

Tools: Continuous Integration

- Some tests need to be manual, but as much as possible automate.
- Systems such as Jenkins allow you to automatically check-out, build and test the system and then inform people if the build is still working.
- If this is done very night, you don’t have the last-minute scenario of everything broke as you try to integrate all the pieces.

Other Tools

- Much UI tasting can be automated using tools such as Selenium.
- Debuggers let you look at live code, step through and see why it breaks.
- Profilers are like debuggers for performance problems.

More iteration

- Introducing:
    - Test Driven Development (TDD)
    - Refactoring.

Summary

- Testing cuts across the entire life cycle: requirements, design, development and maintenance.
- It is crucial to delivering quality software.
- Testing is done at different levels and for different reasons.
- There are tools that can help, especially when trying to automate as much testing as possible.

Moving on from here …

- In the practical this week, we’ll be doing more with JUnit.
- In the group project, try to polish off the requirements this week but also have a “definition of done” that can be used for validation.

University of Strathclyde Glasgow