Putting Testing First

CS 4501 / 6501
Software Testing

[Ammann and Offutt, “Introduction to Software Testing”]
Software Crisis

• Programmers just programmed
  • Result was ad hoc structure
  • Code eventually became hard to maintain
  • Managing complexity was challenging

• The tendency towards irreducible number of errors

• Most software development faced
  • Overdue schedule
  • Exceeding initial budget
  • Inadequate software quality
  • High software maintenance cost
Traditional Cost-of-Change Curve

- Traditional software development methods
  - Focus: extensive modeling and up front analysis
  - Goal: reveal problems and changes as early as possible

More work must be revised
Root problem is harder to find

[AO, p.55]
Increased Emphasis on Testing

If high-quality testing is not centrally and deeply embedded in your development process, your project is at high risk for failure

• Fail in the technical sense
  • Lose control of what the code actually does

• Fail in the business sense
  • Your competitors roll out better functionality faster
Traditional Assumptions

1. Modeling and analysis can identify potential problems early in development

2. Saving implied by the cost-of-change curve justify the cost of modeling and analysis over the life of the project

- These assumptions are true if the requirements are always complete and current
- In reality, customers keep changing their mind
- Changes reflect the requirements
Supporting Evolutionary Design

• Traditional design advice says to anticipate changes
• Designers often anticipate changes that don’t happen
• Both anticipated and unanticipated changes affect design

[AO, Agile Test]
Agile Methods

• Agile methods start by recognizing that neither assumption is valid for many current software projects
  • Software engineers are not good at developing requirements
  • We do not anticipate many changes
  • Many of the changes we do anticipate are not needed

• Requirements (and other non-executable artifacts) tend to go out of date very quickly
  • We seldom take time to update them
  • Many current software projects change continuously

[AO, Agile Test]
Defect Discovery: Traditional vs. Agile

Cost

Length of Feedback Cycle

Programming defect found via Pair Programming

Programming defect found via Continuous Integration

Design or programming defect found via Test Driven Development (TDD)

Requirements or design defect found via Active Stakeholder Participation

Requirements or design defect found via Model Storming

Defect found via independent parallel testing

Requirements defect found via traditional acceptance testing

Design defect found via traditional system testing

Programing defect found via traditional system testing

Defect found via a review or inspection

Copyright 2006-2009 Scott W. Ambler
Managing the Cost Curve

• Test harness as guardian
  • (Near) Instant feedback on changes (or mistakes)
    • An hour? Ten minutes? Less?
  • Something is executable from the very beginning

• Role of continuous integration
  • Effective communication mechanisms

• De-emphasize non executable artifacts
  • If it doesn’t execute, it’s not checkable

• Avoid anticipating future needs
  • YAGNI: You Ain’t Gonna Need It
Test Harness as Guardian

• What is correctness?
  • Traditional: universal
  • Agile: existential

• Limit view of correctness
  • Traditional: define all correct behavior completely at the beginning
  • Agile: define correctness of some behavior with specific tests
    • If the software behaves correctly on the tests, it is correct

Even as the software (including the test cases) evolve, the correctness of the system at any single point in time is subject to immediate verification by running the test set.
Test Harness Verify Correctness

- Tests must be **automated**
- Every test must include a **test oracle** (mechanism that can evaluate whether that test passes or fails)
- Tests (executable artifacts) replace the **requirements** (non-executable artifacts)
- Tests must be **high quality** and must run **quickly**
- Tests must be run **every time** changes are made to the software

Test harness runs all automated tests efficiently and reports results to the developers
Activity: Agile Airplane

Air shows have become popular spectator events. Air shows across the country have placed orders for planes, we need to fulfill them. Everyone here is part of the “Awesome Agile Aviation” Company and can coordinate in any way.

• Roles:
  • QA/testers: Need 7 QAs/testers to evaluate the product after each Sprint
  • Developers: Form 7 teams. Each team has $40 in the bank.

• Economics: The company has a fixed cost (burn rate) of $12 per team per Sprint, and revenue from accepted orders

• Coverage criteria: “Done” criteria

• Test harness: appearance matches the picture, color coding in place, fly 6ft (or 2m) over table

• 3 Sprints. After each Sprint, QA/testers will evaluate the airplanes each team produces – “accept” or “reject”

[Adapted from Agile Airplane Game, by John Heintz, GistLabs]
Agile Airplane – Sprint 1

Sprint 1 (5 minutes)

- Developers:
  - Use the provided supplies, follow the production instruction#1 to produce airplanes
  - You have 1 minute to plan Sprint
  - You have 4 minutes to run Sprint

End of Sprint 1 (2-4 minutes)

- Developers:
  - Demonstrate to QA/testers

- QA/testers:
  - If the airplanes pass the tests, accept the planes, group batches of “done” planes
  - If the airplanes fail the tests, tell the teams about acceptance criteria and reject the planes (“rip” them)
  - Pay $20 for every complete delivery of 15 airplanes (i.e., add $20 to the team financial chart)
  - Update the team financial chart with $12 burn rate
Agile Airplane – Sprint 2

**Sprint 2 (5 minutes)**

- Developers:
  - Use the provided supplies, follow the production instruction#2 to produce airplanes
  - You have 1 minute to reflect and plan
  - You have 4 minutes to run Sprint

**End of Sprint 2 (2-4 minutes)**

- Developers:
  - Demonstrate to QA/testers

- QA/testers:
  - If the airplanes pass the tests, accept the planes, group batches of “done” planes
  - If the airplanes fail the tests, tell the teams about acceptance criteria and reject the planes (“rip” them)
  - Pay $30 for every complete delivery of 10 airplanes (i.e., add $30 to the team financial chart)
  - Update the team financial chart with $12 burn rate
Sprint 3: High value / High risk (5 minutes)

• Developers:
  • Use the provided supplies, follow the production instruction#3 to produce airplanes
  • You have 1 minute to reflect and plan
  • You have 4 minutes to run Sprint

End of Sprint 3 (2-4 minutes)

• Developers:
  • Demonstrate to QA/testers

• QA/testers:
  • If the airplanes pass the tests, accept the planes, group batches of “done” planes
  • If the airplanes fail the tests, tell the teams about acceptance criteria and reject the planes (“rip” them)
  • Pay $40 for every complete delivery of 5 airplanes (i.e., add $40 to the team financial chart)
  • Update the team financial chart with $12 burn rate
Activity: Wrap-up

- What aspects were applied?
- How were test harnesses used?
- How were criteria applied?
- How were acceptance tests applied?
- How likely had the quality been improved from Sprint to Sprint?
Testing as Central Activity

AGILE METHODOLOGY

[Image from http://www.twilightsoftwares.com]
Testing as Central Activity: TDD

[More TDD and exercise .. Later]
Continuous Integration

• Agile methods work best when the current version of the software can be run against all tests at all time
Continuous Integration

- A Continuous integration server rebuilds the system, returns, and re-verifies tests whenever any update is checked into the repository.

Mistakes are caught earlier

Other developers are aware of changes early

The rebuild and re-verify must happen as soon as possible (tests need to execute quickly)

Modified system is still correct (change detected earlier)

Team leaders, testers, managers, clients etc notify

Feedback

Set status

CI server

Report results

Static analysis

Build servers

Check out

Check in
System Tests in Agile Methods

• Traditional testers often design system tests from requirements

• What if there are no traditional requirement documents?
User Stories

- A few sentences that captures what a user will do with the software
  - In the language of the *end user*
  - Usually small in scale with *few details*
  - Not archived

- Withdraw money from checking account
- Support technician sees customer’s history on demand
- Agent sees a list of today’s interview applicants
Acceptance Tests in Agile Methods

User Story

Acceptance Test (Failing)

TDD Test 1

Change software & Refactor

Tests archived

Acceptance Test (Passing)

Continue adding TDD tests until acceptance test passes

Change software & Refactor

Refactoring avoids maintenance debt

TDD Test 2

[AO, p.60]
Wrap-up

• More companies are putting testing first
• This can decrease cost and increase quality
• The definition of “Correctness” becomes restricted but practical
• We embrace evolutionary design
• We use test harness as guardian
• To facilitate the testing process, sometimes we need test doubles.

What’s Next?
• Test automation