Lessons Learned in Teaching Test-Driven Development

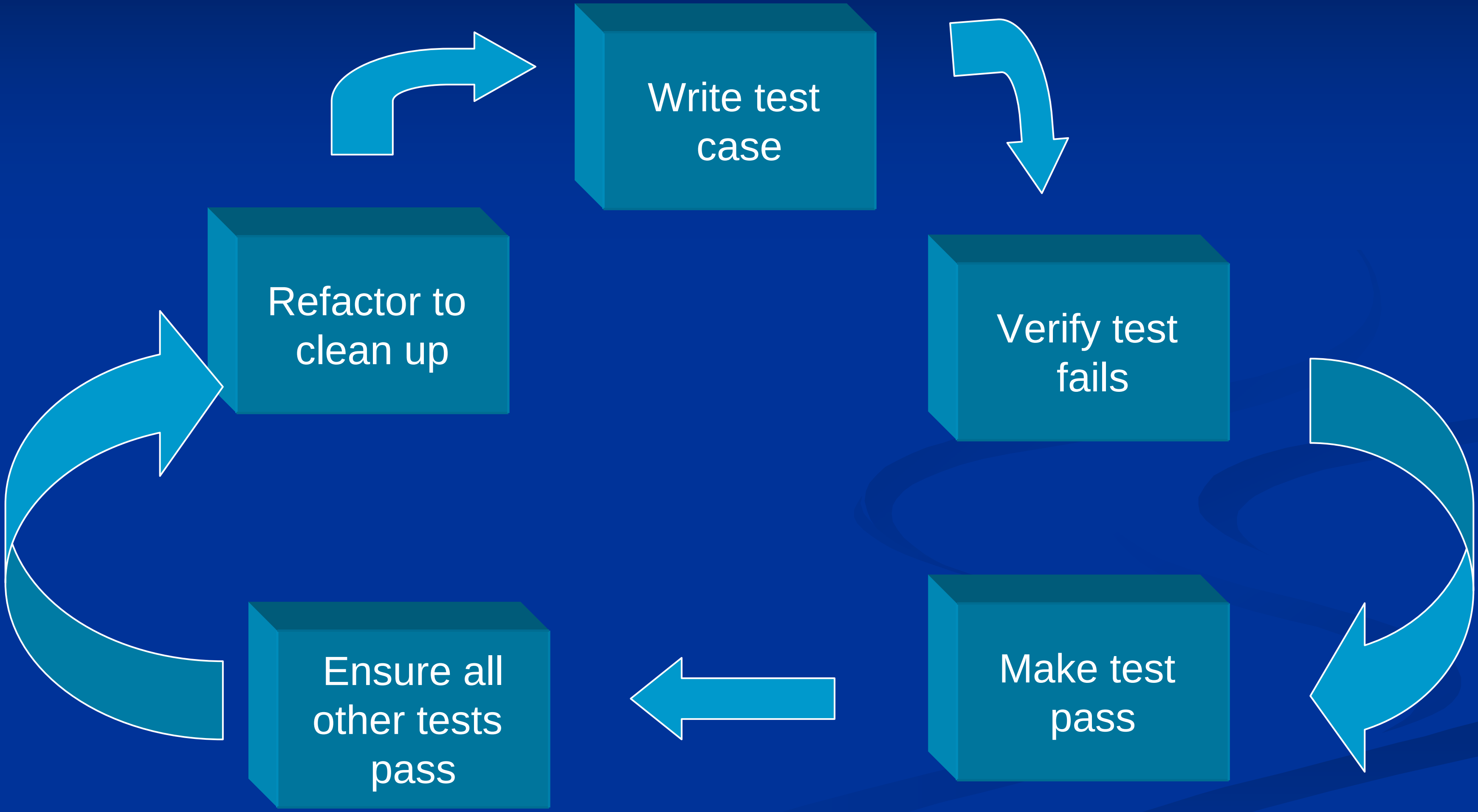
Andy Tinkham

Florida Institute of Technology

About Me

- Developer
  - Worked on naval simulation used by Taiwanese government and JPL
- Tester
  - 8 years of automated testing prior to graduate school
- Doctoral Student
  - 4 years and counting
  - Studying with Cem Kaner at Florida Tech
  - Dissertation topic: Creating a tester’s toolkit and then experimentally determining the effects on testers

Test-Driven Development (TDD)



Write test case

Verify test fails

Make test pass

Ensure all other tests pass

Refactor to clean up

Florida Tech Courses

- Testing 2
- Intro to Java Programming
- Testing Tools

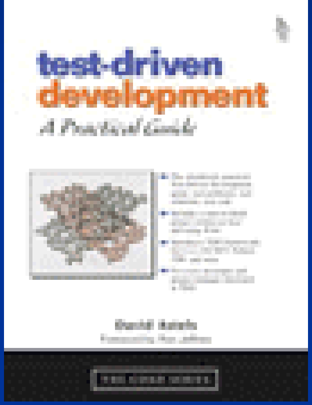
Testing 2

- Second course in sequence (first is blackbox software testing)
- Covers unit testing, TDD, and some scripting for testers (in Ruby)
- Offered 4 times so far

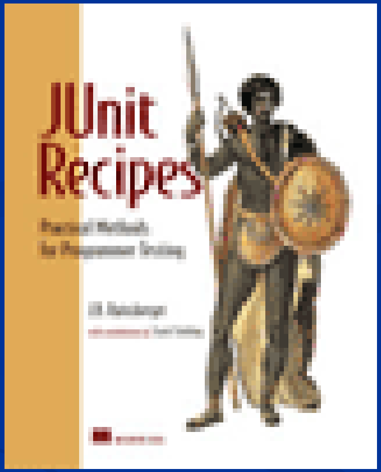
Testing 2 students

- Generally 9-13 students per session
- Predominantly undergraduates, but 3-4 graduates each time
- Course is required for B.S. in Software Engineering, and optional for other degrees

Testing 2 Texts



Test-Driven Development: A Practical Guide, by Dave Astels



JUnit Recipes, by J. B. Rainsberger

Software Quality Engineering, by Jeff Tian

Testing 2 Structure

- Begins lecture-based to cover core concepts
- Becomes project-based ~ 1/3 of the way through semester
- 2 projects
  - New development using TDD
  - Test-driven maintenance
- 2 exams
  - Concept-based midterm
  - Project based final (in Ruby) (see http://blackbox.cs.fit.edu/blog/kaner/archives/000008.html)

Testing 2 Strengths

- Projects closer to real world size than "toys"
- Highlighting differences between new development and maintenance
- Increased confidence for students

Testing 2 Difficulties

- Possible for students to write code for assignment, then go back and write tests
  - Generally easily spotted
  - Require documented iterations for each submission showing a single cycle
- Paired programming helpful but hard to do in classroom

Testing 2 Lessons Learned

- Starting on toy problems makes TDD just seem like extra work and biases students against the concept
- The more programming experience someone has, the harder the transition to a TDD mindset seems to be
- Students seemed to like doing the maintenance project as a class

Intro to Java (CSE 1001)

- First programming course taken by most incoming first-year students
- Offered once with TDD so far
- Covers the basics of computers and programming in Java

•Data types

•Classes

•Constructors

•Public vs. Private methods/fields

•Strings

•Constants

•Layered architectures

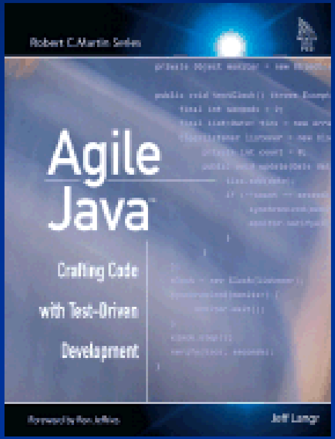
•Arrays

•Recursion

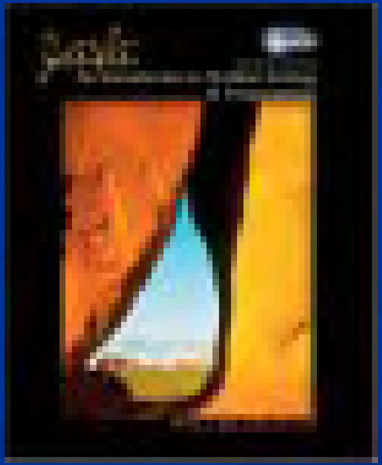
CSE 1001 Students

- Predominantly first-year undergrads who had done poorly in CSE 1001 in the fall 2005 semester
  - Got a 'D' or an 'F'
  - Withdrew in poor standing
- Several students had learning disabilities
- Also had one 12-year old home schooler

CSE 1001 Texts



Agile Java, by Jeff Langr



Java: An Introduction to Problem Solving and Programming (4th ed.), by Walter Savitch

CSE 1001 Structure

- Class was largely project based
- Lectures as needed to explain concepts
- Towards end of semester, instructors began pairing with individual students during class time
- 3 midterm exams (mixed conceptual & project)
- 1 final (project)

CSE 1001 Strengths

- Pairing with the students made a huge difference to many students
- JUnit very useful as an exploration tool

CSE 1001 Difficulties

- With no testing background (and little development background), students had a harder time understanding the concept of a unit test
- Text books not ideal
- More students seemed to try to write the code then write the tests afterwards

CSE 1001 Lessons Learned

- JUnit makes a great tool for exploring the language's capabilities
- Pairing with individual students is critical
- Need to spend a lot of time on test design to get the concept across (pairing might have alleviated some of this had it been done earlier)
- TDD helped provide structure for students to tackle task incrementally

Testing Tools

- 3 pairs of students
  - 3 graduate students
  - 3 undergraduates
  - 5 of the students had already taken Testing 2
- Each pair worked on own development using Agitator

Testing Tools Lessons Learned

- Making the mental leap to using JUnit isn’t enough
- Real projects make it much easier to learn tools

Lessons Summary

- Non-trivial projects work much better to illustrate concepts
- Pairing or class-wide projects can make a huge difference in understanding
- Switching to a TDD approach requires a mental shift proportional to the amount of programming experience
- Use unit testing tools when learning a new language to gain deeper understanding

Questions