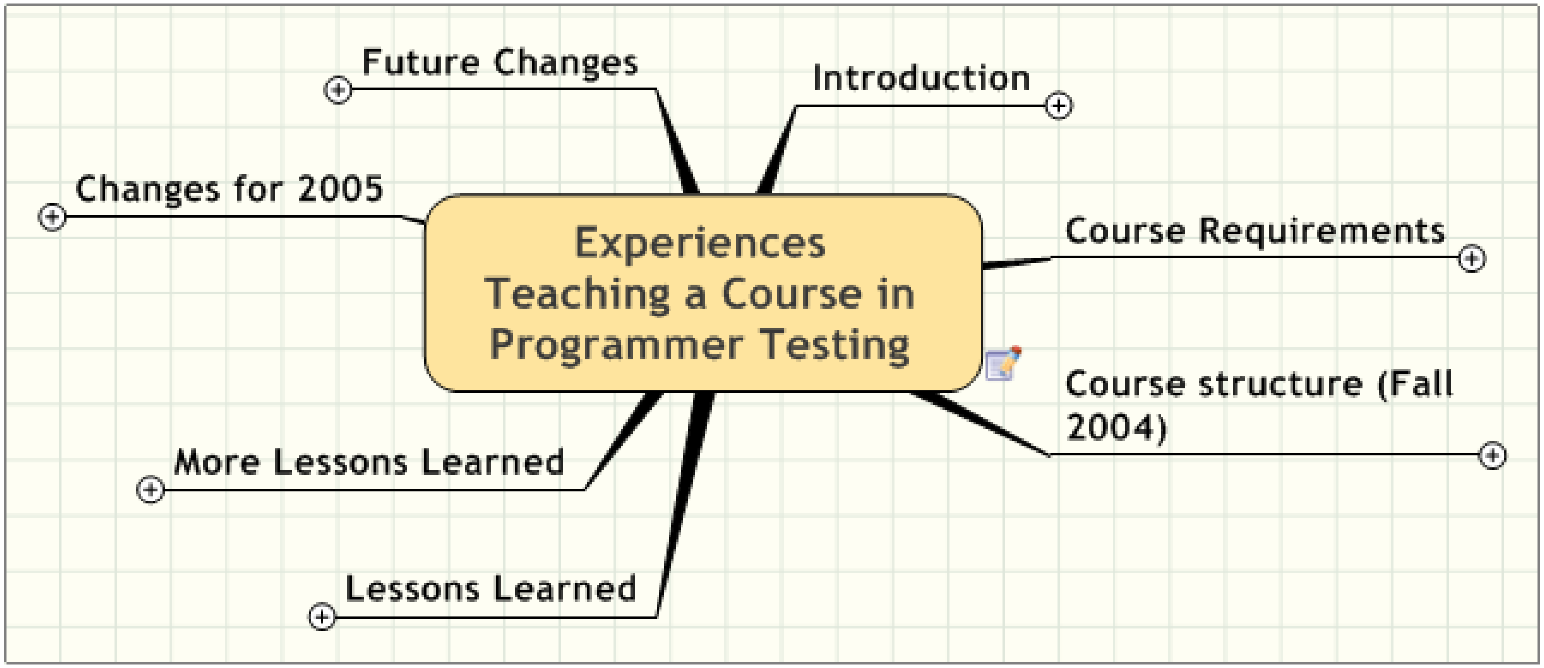
Experiences Teaching a Course in Programmer Testing

Andy Tinkham
andy@tinkham.org
Cem Kaner
kaner@kaner.com

Experiences Teaching a Course in Programmer Testing



Introduction

- Offered 3 times now
- Required for SWE undergrads
  - Junior/Senior level
- Optional for grads
- Taught in computer lab
  - One computer per student
- Ideally 10-16 students per class
- Semester-length course

Course Requirements

- Make better testers
- Improve average programmers
- Develop future project managers
- Create testing toolsmiths & architects
- Give students soft skills practice
- Help students to shine in employment interviews

Make better testers

- Broader perspectives of testing activities
- Increase awareness of opportunities to collaborate with programmers

Improve average programmers

- More thoughtful
- More aware
  - What they're writing
  - Why they're writing it
- More capable of writing code that works

Develop future project managers

- Ways that programmer testing can be better than black-box testng teams
  - More efficient
  - Cheaper

Create testing toolsmiths & architects

- Give students experience creating robust test tools

Give students soft skills practice

- Teamwork
- Presentations

Help students to shine in employment interviews

- Course deliverables as examples of work
  - High-quality deliverables
  - Non-toy examples

Course structure (Fall 2004)

- 12 students
- Lecture- & project-based
- Texts
- Grading
- Project 1
- Project 2
- Mid-term Exam
- Final Exam
- Materials will be available soon on http://www.testingeducation.org/pt

12 students

- 9 undergrad
- 3 grad

Texts

- Astels' Test Driven Development: A Practical Guide
- Hunt & Thomas' Pragmatic Unit Testing in Java with JUnit
- Holzner's Eclipse
- Thomas & Hunt's Programming Ruby: The Pragmatic Programmer's Guide (2nd Ed.)

Grading

- Projects - 40%
- Mid-term Exam - 20%
- Final Exam - 30%
- Homeworks, Quizzes, In-class assignments - 10%

Covered

- TDD with JUnit and Eclipse
- FIT (briefly)
- Ruby & COM Automation

Project 1

- New development with TDD
- Designed to start off following Astels' example, but then diverge
- 4 week project, extended to 6-7 weeks
- Done in pairs

Project 2

- Maintenance of existing code
- Scrapped due to hurricanes
- Replaced with make-up project to avoid drastically harming student's grades
  - Gave students best submission of proj 1 & additional stories to add

Mid-term Exam

- Short-answer questions designed to verify knowledge of basic questions

Final Exam

- Develop test tool in Ruby to compare Microsoft Excel and OpenOffice Calc
- Designed to serve as model for tester creating home-grown tool to accomplish their duties

Lessons Learned

- Need to emphasize distinction of layers for TDD of test tool
- TDD is counterintuitive to many students
  - Students must unlearn & reinterpret prior learning
- JUnit & Eclipse facilitate trial & error study of how commands work
- TDD for new code is very different than TDD on existing code

More Lessons Learned

- Split big problems into smaller chunks
- Using basic algorithms as initial TDD exercises leads to problems
  - Good programmers don't see why taking a longer process to accomplish something basic is good
  - Weak programmers have trouble even implementing the algorithm
- Need difficult enough examples to make process worth it
- Not all CS/SE students can or WANT to program

Changes for 2005

- New texts
  - D'Anjou, et al, Java Developer's Guide to Eclipse
  - Rainsberger's JUnit Recipes
  - Mugridge & Cunningham's FIT for Developing Software
- More focus on test design
- More focus on traditional unit testing techniques
- Split up project 1 into more iterations
- Full second project
- Changes in final exam

Future Changes

- Videotape lectures
- Make class more discussion/activity based