SYLLABUS -- SOFTWARE TESTING 2, FALL 2009

Instructor:	Cem Kaner
Office Phone:	674-7137
Office Address:	Olin Engineering 247
email:	kaner@cs.fit.edu
Skype:	cem.kaner
Office Hours:	Tuesdays 6:15 to 8:00, Thursdays 3:00 to 4:30
Teaching Assistant:	Nawwar Kabbani
TA email:	nkabbani2008@fit.edu
Start Date:	August 18, 2009
Course Credits:	3
Class Times:	5:00 to 6:15 pm, Tuesday & Thursday, Olin EC 228

Final Exam, December 8, 6-8 pm

Required Text:

- Objects First with Java: A Practical Approach Using BlueJ (Barnes & Kolling, 4th edition), Chapters 1-10, 12, parts of 13 and 14
- Several readings (handouts or electronic copies posted in the Moodle course site)

Required Tools:

- Java 1.6
- Eclipse 3.5
- Checkstyle (code style)
- Emma (code coverage)
- JUnit (unit testing)
- Subversion

Learning Objectives:

- Relearn basic programming skills through a lens of writing code that always works -- move from programming to software engineering.
- Adopt industry standard development tools and use them effectively,
- Improve your cognitive skills in basic development, especially problem decomposition and technical (programmer to programmer) communication
- Learn glass box test techniques
- Familiarity with the agile approach to development lifecycles, and gain some experience with agile approaches
- Work through exercises, and develop work products, that I think are likely to help you interview for new positions and do well in your first year on the job

Assessments:

- **Multiple choice quizzes** (open book) -- 10%. I will drop your worst two quizzes.

- **Homework** (several small assignments, to be completed within a few days after assignment) -- 15%. I will drop your worst three homeworks.
- **Assignments** (larger individual or group development projects) -- 30%. You must do all three assignments. (Note: I may drop this to two assignments worth 10% each and add more homeworks, raising the homework percentage to 25% from 15%)
- **Midterm exam** (closed-book in-class exam) -- 15%
- **Final exam** (take-home programming project) -- 30%
- **Bonus tasks**: In the first two weeks of the course, I will suggest some topics that I would like students to research and use to create supplementary material for the course that will help the other students or give a presentation to the class. If you want to take on one of these tasks, you must sign up for it by September 17. I will not accept late sign-ups because my experience is that students who pick up bonus tasks late in the term do a poor job with them. The bonus task will add between -5 (minus five--for example if you sign up for a task but then don't do it, or if you plagiarize the work) and +10 points.

CATALOG DESCRIPTION:

SOFTWARE TESTING 2. Explores structural (glass box) methods for testing software. Testing of variables in simultaneous and sequential combinations, application programmer interfaces, protocols, design by contract, coverage analysis, testability, diagnostics, asserts and other methods to expose errors, regression test frameworks, test-first programming.

DESCRIPTION FOR THIS TERM:

Florida Tech is widely known for its test-related education. At conferences of testing practitioners, Florida Tech has often been described as the best school for testing in the United States. The Testing 2 course has been part of the growth of this reputation. It integrates testing and programming knowledge and skills and the students in the course do "real" work, creating work products that employers find impressive.

We focus on programmer-testing, that is, on the types of tests that programmers create to test their own code. The approach to programmer-testing that we follow is called Test-Driven Development (or Test-Driven Programming). In TDD, the programmer creates the program in small increments. Each increment looks like this:

- decide what functionality you are going to add to the program
- write a test to check whether the functionality was created correctly
- run the test to see what the failure looks like and to confirm that the test will in fact fail if the functionality is not present
- write the code
- run the test (and all the other existing unit tests for the program under development)
- fix the code
- run the test (repeat test/fix/test/fix until the code finally works)
- refactor the code (reorganize working code to make it more maintainable)

- rerun the tests (fix and retest if necessary)
- save the revised code to the source control system

The tools are easy to learn and use. Sometimes a little tedious, but easy.

Technical challenges: Many students face technical challenges in learning this approach:

- **decomposing programs into small tasks (the increments).** This is a critical, basic programming skill. To our astonishment over the past 7 years, graduate students have found this much harder to learn (or to adopt) than undergraduates (instructors who teach TDD at other universities tell us their graduate students have similar problems). The large majority of graduate students who have taken this course have dropped out or failed the course.
- **designing useful tests.** Most books on the current generation of unit test tools focus on how to create and maintain the tests, spending relatively little time on what tests to create.

Psychological or tactical challenges: Some students have treated this course as overwhelming and adopted unsuccessful tactics for dealing with it. All three instructors who have taught the course have seen similar problems. If you fall into these traps, you should come to Nawwar or me for help, or you should seriously consider dropping the course:

- ***Some students are simply unwilling to program in a test-driven development (TDD) style.*** TDD is not for everyone, but it's what we are teaching in this course. If you're going to learn anything in this course, you have to give TDD a fair chance. The key thing to realize is that, no matter what task we are doing, we are doing it to illustrate test-driven development, to extend your skills in test-driven development, and if you are not doing it that way, you are missing the point of the exercise. If the program "works" perfectly, but wasn't done via TDD, it didn't achieve the task that was assigned.
- ***Some students try to fool us with shortcuts.*** For example, students write their code (or copy code fragments from the Net), then add a few unit tests after the fact. ***Don't do that.*** You'll do at least as well if you submit an incomplete program that is well tested than a more complete program that lacks most of its unit tests. Some students go further: They sometimes dummy up some phony tests to make it look as if that part of the program had been tested and passed. ***This is dishonest--it misrepresents the state of your code.*** If Nawwar or I see this in code that we read, we will stop reading and zero the assignment. If we see this in the final exam, we will zero the exam.

STRUCTURE OF THE COURSE

We follow two parallel streams

- Lecture-driven tour of test-driven programming and test design
- Textbook-driven review of Java programming, applying TDD and good test design to the material

We assign tasks (homework, quizzes, assignments) from each stream

- exercises from the textbook
- exercises based on common interview questions for programming jobs
- exercises based on issues raised in the lectures

Tentative list of lecture topics

- Overview of software development
- Programmer testing. Basics of jUnit
- Refactoring (1)
- Problem decomposition
- Source control systems, using Subversion
- Testing control structures (1)
- Basics of structural coverage
- Testing data handling (simple inputs, combinations, results)
- Testing Boolean expressions
- Testing lists and sorts
- Exception handling and testing exceptions
- Assertions and probes
- Testing control structures (2), with exceptions, assertions, and probes
- Communicating the intent of your software
- Generating random numbers
- Using simulations in testing
- Using algorithms and sample code from the web
- Source code analyzers

Lecture Topic List Updated October 2009

Topic	Total	Done	To Come
Overview of software development	1	1	
Programmer testing. Basics of jUnit	1	1	
Refactoring	3	3	
Problem decomposition	1	1	
Source control systems, using Subversion	1	1	
Testing control structures	3	2	1
Basics of structural coverage	1	1	
Testing basic data structures	3	2	1
Testing lists and sorts	1	1	
Exception handling and designing tests that include exception-handling	1	1	
Parameterized test cases	1	1	
Creating test files for higher volume processing of test cases and expected results	1	1	
Generating random numbers; applying randomness in software testing	6	5	1
Testing data handling (simple inputs, combinations, results)	1		1
Testing Boolean expressions	3		3
Using random perturbation of test cases to create higher-volume tests	1		1
Developing oracles for automated tests	2		2
Using simulations in testing	1		1
Applying what we know to classic programming problems	4		4
Using algorithms and sample code from the web	1.5		1.5
Working with assertions and probes	3		3

GRADING POLICIES

- I will not accept late quizzes or late homework. We want to grade these quickly and get them back into your hands quickly--delaying for tardy students makes that much harder. I know that everyone has critically busy schedules or personal challenges at some point that would normally force you to hand stuff in late. To help you cope with these, in this course, you can drop a couple of quizzes and three homeworks.

- Late assignments lose one letter grade (10%) per day. I will not accept late assignments more than 5 days late.
- I will not accept late exams.
- Most assignments (including homework and exam questions) require you to use test-driven development. You must submit your tests with your code. Grading of the code includes grading of the tests.
    - Unless the assignment specifies otherwise, code that does not ***demonstrate*** the application of TDD will get a failing grade (0)
    - If the code doesn't compile successfully, I will assign a grade of zero.
    - If there are no unit tests, I will assign a grade of zero.
    - If the code fails any of its unit tests will I will normally assign a grade of zero. ***Exception***: If you can't get the code to pass a test (e.g., you run out of time before the assignment is due and it still has a bug), include a note with your assignment that describes the failure and the steps you have taken to try to debug it. We will grade that bug gently, and if your debugging strategy is good, we will award points for it.
- To pass the course, you must have a passing grade on the exams. The exams together are worth 45% of your grade. To pass CSE 4415, your total exam score must be 27/45 (D). To pass SWE 5415, your total exam score must be 31.5 (C).

ACADEMIC INTEGRITY

- Homework and Assignments
    - You are welcome to use ideas, algorithms, or even code from books or the web. ***If you use someone else's work, you must reference it. This is true whether you are using work from a web page, the course textbook, or some other source. Failure to give credit to your sources is plagiarism and will be handled according to the University's Academic Integrity policies.***
    - You may consult other students (members of this class and students not currently enrolled) when preparing an assignment. Please give credit to each helper by including in the assignment the person's name and a brief description of how he or she helped you. If you are copying any code from her or him, specifically identify that code. ***If you receive help but submit work that fails to acknowledge your helpers, I will zero your paper and may take additional action in accordance with the University's academic integrity policy.***
    - You are welcome to do quizzes, homework, and assignments in pairs. If you write homework or an assignment together, submit it as one submission and make both of your names very visible.
        - If you work with the same person several times, I will ask you to select a different partner for your next tasks. Over the term, I'd like you to pair up with at least three people.
- Quizzes
    - Quizzes are open book
- Midterm exam

    - The usual rules governing cheating in closed book exams apply to the midterm.

- Final exam
    - You are welcome to consult books or published material on the web. You may not get help from a live human, whether they are members of this class or not.
    - As with homework and assignments, if your exam uses someone else's code or ideas, you must reference them. Failure to give credit to your sources is plagiarism.