CSC323 2010S *Software Design*

# Class 10: Unit Testing

**Held:** Thursday, February 25, 2010

**Summary:** We consider the related processes of unit testing and test-driven development.

**Related Pages:**

- EBoard.

**Notes:**

- Readings for Monday: Skim Chapters 1 and 2 of *Version Control with Subversion* (for Subversion 1.5); Read BC 2: Subversion's Delta Editor.
- At times, I felt like I read something different than the rest of you.
- EC: Today's CS Extra.
- EC: Tomorrow's CS Table.

**Overview:**

- Project Discussion.
- About Unit Testing and Test-Driven Development.
- Beautiful Testing.
- PyUnit: Unit Testing in Python.
- Lab: Unit Testing.

# Project Ideas

- We'll listen to your project ideas and consider which might be appropriate for the class.
- Remember: A good project involves natural polymorphism, inheritance, and encapsulation.

# Unit Testing and Test-Driven Development

- Unit Testing and Test-Driven Development are, as their names suggest, test-centric development methodologies (or components thereof).
- Unit Testing: Write tests for each "unit" of the program. (A unit is typically a small piece of the program.)
- Test-Driven Development: A cyclic strategy in which we write tests before adding new features.
- We've already talked about the import of having an automated test, one which simply reports success or a list of failures.

Why write tests?

- Gives us more confidence in the code.
- Helps us think about the problem.
- Gives us more confidence to *change* our code.
- ...

Why write tests *first*?

- Evidence suggests that if we don't write the tests first, we will be less likely to write the tests.
- As is the case for writing documentation, when we write the tests first, we think more carefully about the problem.
- When we write tests first, we are less likely to be biased by the design of the procedure we wrote.
- When we write tests first, we
- ...

Because of the power of unit testing, almost every modern language has a unit testing framework.

- Many of them (including JUnit and PyUnit) are based on Kent Beck's original unit testing design.
- They take advantage of important OO techniques, such as inheritance and introspection.
- The basic of this is a test case.
  - Set up some data for testing (e.g., build an array)
  - Run a series of tests on the data
  - Clean up after yourself, if necessary.
- Test cases are then grouped hierarchially into suites

# Beautiful Testing

- There were a host of questions, which we'll try to cover.
- We'll also look at the normal
  - What is beautiful?
  - What is ugly?
  - What can you take away from this all?

# Unit Testing in Python

- A variant of the original Beck-style testing framework.
- We use the `unittest` package.
- Individual test cases subclass `unittest.TestCase`.
- Suites are built from `unitTest.TestSuite`.
- We run suites from the interactive pane with `unittest.TextTestRunner().run(`*`suite`*`)`.

# An Exercise

- Your goal: Design two test suites
  - One for a binary search.
  - One for a rational number class.
- We'll discuss them as a group.
- If we don't have enough time, we'll turn them into an assignment.

---