CSC161 2010F *Imperative Problem Solving*

Class 24: Unit Testing

Held: Wednesday, 6 October 2010

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

Related Pages:

- EBoard.
- Reading: Unit Testing 101 for Non-Programmers.

Notes:

- EC for tomorrow's cool CS extra: Shitanshu on his work at Amazon (or at least the parts he's allowed to tell you about.
- No reading for Friday!
- Reminder: SUBMIT QUESTIONS!
- Checking: Do you know where to find the sample code we build each day?
- Today is a recitation class.

Overview:

- Unit Testing.
- Unit Testing Frameworks.
- Questions.
- Example: Average.
- Example: Binary Search.

Unit Testing Basics

- Unit Testing: An approach to testing your program.
- Follows the scientific method (more or less): Make a hypothesis about what your program should produce and check the result.
  - Done multiple times.
- We test each "logical unit" of the program separately.
- If the units don't work correctly, the whole won't work correctly.
- Write tests *before* you write the code.
  - Tests remind you what you want your code to do.
  - You're more likely to write tests if you write them first.
- Each time you modify your code, rerun the tests to make sure you haven't broken anything.

Unit Testing Frameworks

- Unit testing frameworks make it easier to do unit testing.
- Essentially, you mix tests in with your code and the framework can find those tests.
- As importantly, frameworks let you combine different unit tests.
  - You can check whether a change made in one unit breaks another unit.
  - If the program is well designed, the change should not affect another unit, but it might.

Questions

- Unit testing seems to focus on a bottom-up development methodology. Is that really the case?
- How do programmers break up the testing?
- Is this like building a lot of helper procedures and testing them all individually instead of just testing the main function at the end?
- Do we test all possible/probable inputs?
- What is “continuous integration”?
- What do unit tests look like?

Exercise: Average

- We’ll start with an interactive tester.
- We’ll continue with a hard-coded tester.
- We’ll find some errors and attempt to correct our code.
- We’ll write new code and continue to test.
- We’ll improve our hard-coded tester to give more useful results.

Exercise: Binary Search

- We’ll start with an interactive tester.
- We’ll continue with a hard-coded tester.
- We’ll look for ways to improve the hard-coded tester.
- We’ll improve our hard-coded tester to give more useful results.
