Paul English

CS 2420-003

Robert Baird

January 29, 2013

**Reflection - Unit Testing**

Sometimes it's difficult to actually drive development with thorough testing. Actually, it's always difficult, I've found I rarely get an adequate specification of what I'm attempting to build, let alone a test plan. It usually falls on my own hands to take care of the test plan, and setting up test cases that cover edge cases. I've never written up a formal test plan, though I have gotten relatively stalwart about keeping tests up to date.

The more you practice testing your code at the unit level, oftentimes the better you get at architecting as well. To make a project testable, it usually needs some level of loose coupling that allows you to observe the end-to-end interactions of a component or class. Java does help in providing obvious points of test integration. Other languages may not be as organized, though they are all testable.

The process of writing code requires a developer to continuely run ad-hoc tests, so formal/automated tests are really just a way of saving work, and allowing it to be played back at a later date, once you feel like augmenting things. It's indespensible in any non-trivial project and the only way to guarantee observed bugs are fixed long-term.

Tests don't just need to be relegated to the realm of valid function responses and ensuring the right exception is or isn't thrown. A mature test suite may guarantee the efficiency of certain important functionality, or it might even be able to simulate a user with high-level interaction using some kind of integration test.

Working with JUnit feels like Java, it is Java, so I can't say I have much to comment about it. I tend to prefer concise and terse programming platforms, especially when it comes to testing. With that said, if it works, it works.