Test-Driven Development (TDD)

Chung-Kil Hur

(Credit: Byung-Gon Chun & Many Slides from UCB CS169 taught by Armando Fox, David Patterson)

SWPP, CSE, SNU

BDD+TDD: The Big Picture

- Behavior-driven **design** (BDD)
  - develop user stories *(the features you wish you had)* to describe how app will work
  - via Cucumber, user stories become *acceptance tests* and *integration tests*
- Test-driven **development** (TDD)
  - *step definitions* for new story may require new code to be written
  - TDD says: write unit & functional tests for that code *first,* ***before*** the code itself
  - that is: write tests for *the code you wish you had*

TDD/Testing Key Ideas

- TDD Starting from the acceptance and integration tests derived from User stories, write failing unit tests that test the nonexistent code you wish you had. (RSpec)
- TDD Write just enough code to pass one test and look for opportunities for refactor the code before continuing with the next test. (Red-Green-Refactor)
- Use mocks and stubs in your tests to isolate the behavior of the code you're testing from the behavior of other classes or methods on which it depends.
- Various code coverage metrics help you determine which parts of your code need more testing.

Testing

- Verification – building the thing right via software testing as part of the Agile lifecycle
- We focus on testing, since it doesn't get much attention as other parts of the software lifecycle.
- Approaching software construction from a test-centric perspective often improves the software's readability and maintainability
  - Testable code tends to be good code, and vice versa

Testing

- Exhaustive testing infeasible
- Divide and conquer: perform different tests at different phases of SW development
  - Upper level doesn't redo tests of lower level

System or acceptance test: integrated program meets its specifications
Integration test: interfaces between units have consistent assumptions, communicate correctly
Module or functional test: across individual units
Unit test: single method does what was expected

Testing

- Test Coverage: % of code paths tested
- Regression testing: automatically rerun old tests so changes don't break what used to work
- Continuous Integration (CI) testing: continuous regression testing on each code check-in vs. later testing phase

TDD Outline

- FIRST
- RSpec
- The TDD Cycle: Red–Green–Refactor
- Seams
- Expectations
- Mocks and Stubs
- Fixtures and Factories
- TDD for the Model & Stubbing the Internet
- Coverage, Unit vs. Integration Tests
- Other Testing Concepts; Testing vs. Debugging

FIRST, TDD, and Getting Started With RSpec

QA in Agile

- Antiquated for SaaS apps: Quality assurance is the responsibility of a separate group rather than the result of a good process
- Developers bear far more responsibility for testing their own code and participating in reviews
- QA engineers have largely shifted to improving the testing tools infrastructure, helping developers make their code more testable, and verifying that customer-reported bugs reproducible

Testing Today

- Far more automated
- Tests are self-checking
  - The test code itself can determine if the code being tested works or not
- A high degree of automation is key to supporting the five principles for creating good tests

Unit tests should be FIRST

- **F**ast
- **I**ndependent
- **R**epeatable
- **S**elf-checking
- **T**imely

Unit tests should be FIRST

- **F**ast: run (subset of) tests quickly (since you'll be running them *all the time*)
- **I**ndependent: no tests depend on others, so can run *any subset* in *any order*
- **R**epeatable: run N times, get same result (to help isolate bugs and enable automation)
- **S**elf-checking: test can *automatically* detect if passed (*no human checking* of output)
- **T**imely: written about the same time as code under test (with TDD, written *first!*)

Tools, tools, tools, …

- BDD
  - Cucumber: cucumber works with Ruby, Java, Python, .NET, Javascript, Scala, C++, Lua, Flex or web applications written in language
  - Behave, Lettuce, Freshen for Python
  - cucumber.js for node.js
  - …
- TDD
  - RSpec
  - JUnit, JRuby + maven-rspec-plugin for Java
  - nose, unittest for Python
  - Mocha, should.js for node.js
  - Jasmine, jasmine-node for node.js
  - …
- UI testing automation
  - Capybara, Selenium, Robotium, …

(Conventional) Unit Testing Tool (Used for unit/functional testing)

- xUnit: a framework to write repeatable tests
  - JUnit
  - NUnit
  - CUnit
  - Ruby Test::Unit
  - Python unittest
  - …
- Creating tests: annotation, inheritance, DSL

xUnit

- Creating tests
  - Annotation: Java, C#, …
  - Inheritance: ruby, python, …

- Automatic checking using assertions
  - Tests that exercise happy paths
  - Tests that exercise sad paths

Example: JUnit

(Ref. https://github.com/junit-team/junit/wiki/Getting-started)

Calculator.java

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

CalculatorTest.java

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

Example: A Successful Test

java -cp .:junit-4.XX.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore CalculatorTest

JUnit version 4.12

.

Time: 0,006

OK (1 test)

(you will use a software project management tool, which simplifies running tests.
E.g., make, maven, rake)

Example: A Failing Test

- Replace the line `sum += Integer.valueOf(summand);`

  with `sum -= Integer.valueOf(summand);`

```
java -cp .:junit-4.XX.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore CalculatorTest

JUnit version 4.12
.E
Time: 0,007
There was 1 failure:
1) evaluatesExpression(CalculatorTest)                    <-- Which test failed
java.lang.AssertionError: expected:<6> but was:<-6>       <-- What went wrong
  at org.junit.Assert.fail(Assert.java:88)
  ...

FAILURES!!!
Tests run: 1,  Failures: 1
```