CSC 405, Week 7

Guest Lecture on Entrepreneurship
General Testing Background
Project-Specific Testing Details
Evaluation of Week 7 Critical Tasks

Revised 17 February

I. Weekly Lecture/Lab Overview

A. *Monday*:

1. Guest lecture/discussion on entrepreneurship
2. Brief introduction to testing
3. Lab time for project work

B. ***Wednesday:***

1. Continued testing discussion

2. Testing set up and critical task work in lab

C. ***Friday:***

1. Project-specific testing details
2. Check completion of Week 7 critical tasks
3. Assign Week 8 critical tasks, including testing duties
4. Lab time for project work

II. **Types of testing we'll perform:**

A. ***Functional Model*** **--**

`JUnit` testing through model API

B. ***Functional View*** **--**

`GWTTesting` through view API

C. ***Acceptance, Automated*** **--**

Selenium testing through HCI

D. ***Acceptance, Manual***

-- human testing through HCI

III. Mapping Testing Terminology onto 405 and 406

A. Consider testing activities from Week 3 Lecture Notes.

B. What follows are notes on how they'll be accomplished in 405 & 406.

IV. **Unit Testing**

A. For model, use JUnit3.

B. For view, use GWTTest.

C. For DB layer, adapt SQLite testing framework.

D. For critical methods, formalize with JML.

V. Module Testing

A. For model and view, start with stubbed DB test fixtures.

B. Output results with JUnit Assert methods.

C. Also output to differencable log.

VI. **Integration Testing**

A. Replace stubbed DBs with actual DBs.

B. Re-run module tests.

VII. System Testing

A. Run the top-most model/view tests with Junit Runner.

B. Run automated UI tests with Selenium.

VIII. **Acceptance Testing**

A. Use existing acceptance test procedure.

B. Run for each department's Winter 2012 schedule.

IX. **Black Box Testing**

A. Write initial test cases, using JML specs for critical methods.

B. Particular focus on end-user data validation preconditions.

X. White box

A. Add more tests based algorithm details in code.

B. Add more tests based coverage results.

XI. Testing Design

A. JUnit3 API is well documented.

B. So is GWTTest.

C. And SQLite test framework as well.

XII. **Test Plan**

A. I *strongly* recommend a common, uniform structure for these.

B. Unit test plans in testing method Javadoc comments.

C. Module test plans in class Javadoc comments.

D. Integration test plan as a text document.

XIII. **Top-down Testing**

A. Algorithm with stubbed DBs.

B. Selenium with or without underlying stubs.

XIV. **Bottom-up Testing**

A. Algorithm without UI.

B. Possibly some form of independent DB testing.

XV. **Test Case**

A. Defined specifically by JUnit and GWTTest

XVI. **Testing Oracles**

A. For critical methods, a formal postcondition.

B. For other aspects of test, human-produced validation methods.

XVII. **Testing Stubs**

A. Used primarily for independent algorithm testing.

B. Elsewhere as needed.

XVIII. **Test Driver**

A. Provided by JUnit, GWTTest, Selenium.

XIX. **Regression Testing**

A. With JUnit Assert.

B. Differencable logs for algorithm internals.

XX. **Test Coverage**

A. Use Coberatura.

B. It integrates well with Junit3.

XXI. Test Subsumption

A. Unplug independent DB tests when algorithm is integrated.

B. Much of Junit tests is effectively subsumed by Selenium-driven tests, but we'll do both as part of the official regression suite.

XXII. **Test Automation**

A. Via Eclipse.

B. May also use `ant` builds and/or `make-files`.

XXIII. **Mutation Testing**

A. Might be interesting to do a bit in 406.

XXIV. Testing Harness

A. *Covered above, particularly under Test Drivers.*

XXV. **Testing Frameworks**

A. To summarize:

1. JUnit for model API
2. GWTest for view API
3. Selenium for automated HCI.
4. Human-performed for accepted HCI.

Resources

XXVI. See `405/doc` page.

XXVII. **Code coverage basics.**

A. Measures applied during test execution.

B. Terminology varies quite a bit.

C. List of weakest to strongest follows.

XXVIII. **Code coverage criteria.**

A. Function (method) coverage

B. Statement coverage

C. Branch coverage

Code coverage criteria, cont'd

D. Decision coverage

E. Loop coverage

F. Define-use (d-u) coverage

Code coverage criteria, cont'd

G. All path coverage

H. Exhaustive coverage

XXIX. Common Example

```
public static int f(int i, int j) {
    int k;
    if (i > j) {
        i++;
        j++;
    }
    k = g(i,j);
    if ((k > 0) && (i < 100)) {
        i++;
        j++;
    }
    else {
        i++;
    }
    return i+j+k;
}

static int g(int i, int j) {
    return i-j+1;
}
```

XXX. **Function coverage.**

A. Each function called at least once.

B. Very large-grain measure.

C. Not adequate for final tests.

D. Can be done with one test case for `f`.

XXXI. **Statement coverage.**

A. Every statement is executed at least once.

B. Can be done with two test cases for `f`.

XXXII. **Branch coverage.**

A. The true/false direction of each branch is taken at least once.

B. Requires four test cases for `f`.

XXXIII. **All path coverage**

A. Each distinct control path is traversed.

B. Requires four cases for `f`.

XXXIV. **Decision coverage**

A. The boolean logic of each condition is fully exercised.

B. Requires at least four cases in `f`.

XXXV. **D-u coverage**

A. Every path for every variable to every use of that variable is covered.

B. D-u for i requires three paths in `f`.

C. D-u for j requires two paths in `f`.

XXXVI. When 100% coverage is not attainable.

A. Platform-specific code not "#ifdef'd" out.

B. `Catch`s for exceptions that cannot be forced, e.g., I/O.

C. Uninvoked methods necessary to satisfy `interface` contract.

XXXVII. **Friday In-Class Testing Briefs**

A. Solame meets briefly with Selenium testers.

B. Fisher meets briefly with database, model, and support testers.

XXXVIII. **Practical Matters for JUnit Testing**

A. Make all data members `protected` instead of `private`.

B. At beginning of database-dependent testing, set-up an in-memory database fixture, which persists throughout all test execution.