CSC 309 Lecture Notes Week 8

More on Code Coverage
Acceptance Testing

I. What is code coverage?

A. What's covered during program execution.

B. Typically measured at lines of code.

C. Coverage measure is percentage of *program lines run.*

D. All lines covered => 100%.

II. How code goes "uncovered".

A. Reasons include:

1. *Uninvoked functions*
2. *Untaken conditional branches*
3. *Unexecuted loop bodies*

How code goes uncovered, cont'd

B. During testing, uncovered code means ***there are insufficient test cases.***

III. Coverage Tool Resources

A. See the `309/doc/` page.

B. Note that code coverage is NOT required for Milestone 4, but is for final project.

C. M4 requires ***selection of which coverage tool to use.***

IV. **Where code coverage fits into testing.**

A. Ensure black box tests are adequate.

B. Different levels of coverage exist.

C. Good tests mush ensure
some measure of coverage.

Where code coverage fits into testing, cont'd

D. Coverage measures made during testing

E. Following discussion is of different coverage measures, from weakest to strongest.

V. Code coverage measures.

A. Function (method) coverage.

B. Statement coverage

C. Simple branch coverage

D. Decision branch coverage

Code coverage measures, cont'd

E. Loop coverage

F. Define-use (d-u) coverage

G. All path coverage

H. Exhaustive coverage

VI. A Simple 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;
}
```

```
public static int f(int i, int j) {
    int k;
    if (i > j) {
        i++;          // Block 1
        j++;
    }                 // Block 2 (false)
    k = g(i,j);
    if ((k > 0) && (i < 100)) {
        i++;          // Block 3
        j++;
    }
    else {
        i++;          // Block 4
    }
    return i+j+k;
}

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

A. Possible program paths:

1. B1, B3
2. B1, B4
3. B2, B3
4. B2, B4

VII. **Function coverage.**

A. Each function is 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, g`.

E. Possible path: P1 only

VIII. **Statement coverage.**

A. Every statement is executed at least once.

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

C. Possible paths: P1 and P2

IX. Simple branch coverage.

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

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

C. Possible paths: P1 and P4

X. Decision branch coverage

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

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

C. Possible paths: P1, P3, and P4

XI. D-u coverage

A. Cover every path between var def and use, without intervening def.

B. D-u for i requires three paths in `f`:
   P1, P3, P4

C. D-u for j requires two paths in `f`:
   P1, P3

XII. **All path coverage**

A. Each distinct control path is traversed.

B. Requires four test cases for `f`.

C. Required paths: P1, P3, P3, and P4

XIII. **Details decision branch coverage.**

A. Truth table can help understand

B. E.g., for decision

```
((k > 0) && (i < 100))
```

k > 0	i < 100	(k > 0) && (i < 100)	i	j	Remarks
0	0	0	1	2	i < j means k <= 0
0	1	0	100	101	i < j means k <= 0
1	0	0	100	100	i >= j means k > 0
1	1	1	2	1	i >= j means k > 0

XIV. **Coverage tools.**

A. There are several coverage tools for Java.

B. *Cobertura* is a good one.

C. Example in
`309/examples/cobertura`.

coverage-tools, cont'd

1. Example uses program in these notes.

2. Runs coverage and unit tests

3. There's an `ant` build script in `build.xml`

coverage-tools, cont'd

4. Examples files are:
   - *CoverageExample.java*
   - *CoverageExampleTest.java*
   - *build.xml*
   - *build.properties*

coverage-tools, cont'd

5. Results in:
   - *`reports/cobertura-html`*
     -- the coverage report
   - *`reports/junit-html`*
     -- junit testing report

coverage-tools, cont'd

D. You can modify ant scripts.

E. Alternatively, Coberatura in an IDE.

F. NOTE: Cobertura doesn't support Java 7.

XV. A notable research result on test coverage

A. 2009 paper from Microsoft and Avaya.

B. Observations and conclusions:

Research results on code coverage, cont'd

"Despite dramatic differences between the two industrial projects under study we found that ***code coverage was associated with fewer field failures*** ... . This strongly suggests that code coverage is a sensible and practical measure of test effec-tiveness."

Research results on code coverage, cont'd

"[They found] an ***increase in coverage leads to a proportional decrease in fault potential.***"

Research results on code coverage, cont'd

"Disappointingly *(?)*, there is ***no indication of diminishing returns*** (when an additional increase in coverage brings smaller decrease in fault potential)."

Research results on code coverage, cont'd

"What appears to be even more disappointing, is the finding that additional ***increases in coverage come with exponentially increasing effort***. Therefore, for many projects it may be impractical to achieve complete coverage."

XVI. Data & Method Access During Testing

XVI. Data & Method Access During Testing

A. Must private methods be tested directly?

XVI. Data & Method Access During Testing

A. Must private methods be tested directly?

B. For 309, the answer is ***Yes.***

XVI. Data & Method Access During Testing

A. Must private methods be tested directly?

B. For 309, the answer is ***Yes.***

C. So, use `protected` instead of `private`.

XVI. Data & Method Access During Testing

A. Must private methods be tested directly?

B. For 309, the answer is ***Yes.***

C. So, use `protected` instead of `private`.

D. ***Or*** access `private` by reflection.

XVII. Acceptance testing -- the other kind of functional testing

XVII. Acceptance testing -- the other kind of functional testing

A. Performed by end user.

XVII. Acceptance testing -- the other kind of functional testing

A. Performed by end user.

B. Based on HCI instead of API.

XVII. Acceptance testing -- the other kind of functional testing

A. Performed by end user.

B. Based on HCI instead of API.

C. Similar structure to unit testing.

XVII. Acceptance testing -- the other kind of functional testing

A. Performed by end user.

B. Based on HCI instead of API.

C. Similar structure to unit testing.

D. See the handout.