Path Testing + Coverage

Chapter 9
Assigned reading from Binder

Structural Testing
- Also known as glass/white/open box testing
- A software testing technique whereby explicit knowledge of the internal workings of the item being tested are used to select the test data
- Functional Testing uses program specification
- Structural Testing is based on specific knowledge of the source code to define the test cases and to examine outputs.

Structural Testing
- Structural testing methods are very amenable to:
  - Rigorous definitions
  - Control flow, data flow, coverage criteria
  - Mathematical analysis
  - Graphs, path analysis
  - Precise measurement
  - Metrics, coverage analysis

Program Graph - Definition
- Given a program written in an imperative programming language, its program graph is a directed graph in which nodes are statement fragments, and edges represent flow of control
- A complete statement is also considered a statement fragment

Program Graph - Example

DD-Path
- A decision-to-decision path (DD-Path) is a chain in a program graph such that:
  - Case1: it consists of a single node with indeg=0
  - Case2: it consists of a single node with outdeg=0
  - Case3: it consists of a single node with indeg ≥ 2 or outdeg ≥ 2
  - Case4: it consists of a single node with indeg =1, and outdeg = 1
  - Case5: it is a maximal chain of length ≥ 1
- DD-Paths are also known as segments
DD-Path Graph

- Given a program written in an imperative language, its **DD-Path graph** is a directed graph, in which nodes are DD-Paths of its program graph, and edges represent control flow between successor DD-Paths.
- Also known as **Control Flow Graph**

Control Flow Graph Derivation

- Straightforward process
- Some judgement is required
- The last statement in a segment must be a predicate, a loop control, a break, or a method exit
- Let's try an example...

```java
public int displayLastMsg(int nToPrint) {
    np = 0;
    if ((msgCounter > 0) && (nToPrint > 0)) {
        for (int j = lastMsg; (( j != 0) && (np < nToPrint)); --j) {
            System.out.println(messageBuffer[j]);
            ++np;
        }
    }
    if (np < nToPrint) {
        for (int j = SIZE; ((j != 0) && (np < nToPrint)); --j) {
            System.out.println(messageBuffer[j]);
            ++np;
        }
    }
    return np;
}
```

Control Flow graphs

- Depict which program segments may be followed by others
- A segment is a node in the CFG
- A conditional transfer of control is a **branch** represented by an edge
- An **entry node** (no inbound edges) represents the entry point to a method
- An **exit node** (no outbound edges) represents an exit point of a method

Control flow graphs

- An **entry-exit path** is a path from the entry node to the exit node
- **Path expressions** represent paths as sequences of nodes
- Loops are represented as segments within parentheses followed by an asterisk
- There are 22 different path expressions in our example
Example path expressions

AL
ABL
ABCDGL
ABCDEGL
ABC(DEF)*DGL
ABC(DEF)*DEGL
ABCDGHIL
ABCDGHIL
ABCDGH(IJK)*IL
ABC(DEF)*DEGH(IJK)*IJL

Full list of path expressions in the reading

Code coverage models

- Statement Coverage
- Segment Coverage
- Branch Coverage
- Multiple-Condition Coverage

Statement coverage

- Achieved when all statements in a method have been executed at least once
- A test case that will follow the path expression below will achieve statement coverage in our example
  
  ABC(DEF)*DGH(IJK)*IL

- One test case is enough to achieve statement coverage!

Statement coverage problems

- Predicate may be tested for only one value (misses many bugs)
- Loop bodies may only be iterated once
- Statement coverage can be achieved without branch coverage. Important cases may be missed

```java
String s = null;
if (x != y) s = "Hi";
String s2 = s.substring(1);
```

Segment coverage

- Segment coverage counts segments rather than statements
- May produce drastically different numbers
  - Assume two segments P and Q
  - P has one statement, Q has nine
  - Exercising only one of the segments will give 10% or 90% statement coverage
  - Segment coverage will be 50% in both cases

Branch coverage

- Achieved when every path from a node is executed at least once
- At least one true and one false evaluation for each predicate
- Can be achieved with D+1 paths in a control flow graph with D 2-way branching nodes and no loops
- Even less if there are loops
Branch coverage problems

- Short-circuit evaluation means that many predicates might not be evaluated
- A compound predicate is treated as a single statement. If n clauses, \(2^n\) combinations, but only 2 are tested
- Only a subset of all entry-exit paths is tested
  ```
  if (a == b) x++; 
  if (c == d) x--; 
  ```

Multiple-condition coverage

- All true-false combinations of simple conditions in compound predicates are considered at least once
- A truth table may be necessary
- Not necessarily achievable due to lazy evaluation or mutually exclusive conditions
  ```
  if ((x > 0) && (x < 5)) ... 
  ```

Dealing with Loops

- Loops are highly fault-prone, so they need to be tested carefully
- Simple view: Every loop involves a decision to traverse the loop or not
- A bit better: Boundary value analysis on the index variable
- Nested loops have to be tested separately starting with the innermost

Coverage usefulness

- 100% coverage is never a guarantee of bug-free software
- Coverage reports can
  - point out inadequate test suites
  - suggest the presence of surprises, such as blind spots in the test design
  - Help identify parts of the implementation that require structural testing

Coverage example

- TEX and AWK are widely used programs with comprehensive test suites
- Coverage analysis showed
  ```
  System | Segment | Branch | P-use | C-use |
  ------ | ------- | ------ | ----- | ----- |
  TEX   | 85      | 72     | 53    | 48    |
  AWK   | 70      | 59     | 48    | 55    |
  ```

Is 100% coverage possible?

- Short-circuit evaluation
- Mutually exclusive conditions
  ```
  (x > 2) || (x < 10) 
  ```
- Redundant predicates
  ```
  if (x == 0) do1; else do2; 
  if (x != 0) do3; else do4; 
  ```
- Dead code
  ```
  “This should never happen” 
  ```
How to measure coverage?
- The source code is instrumented
- Depending on the code coverage model, code that writes to a trace file is inserted in every branch, statement etc.
- Most commercial tools measure segment and branch coverage

FAQ about Coverage
- Is 100% coverage the same as exhaustive testing?
- Are branch and path coverage the same?
- Can path coverage be achieved?
- Is every path in a control flow graph testable?
- Is less than 100% coverage acceptable?
- Can I trust a test suite without measuring coverage?
- When can I stop testing?

Some answers...
- When you run out of time
- When continued testing reveals no new faults
- When you cannot think of any new test cases
- When you reach a point of diminishing returns
- When mandated coverage has been attained
- When all faults have been removed

A coverage counter-example
```cpp
void Depository::give_change(int price) {
    int n_100, n_25, n_10, n_5;
    if (deposit <= price) {
        change_due = 0;
    } else {
        change_due = deposit-price;
        n_100 = change_due / 100;
        change_due = change_due - n_100*100;
        n_25 = change_due / 25;
        change_due = change_due - n_25*25;
        n_10 = change_due / 10;
        change_due = change_due - n_10*10;
        n_5 = change_due / 10; // A cut-and-paste bug
    }
}
```