Controlflow- and Dataflow-Based Coverage Criteria & Testing Tools (2)

Professor Mark Gabel
Slides Courtesy of Prof. W. Eric Wong
Department of Computer Science
University of Texas at Dallas

What is coverage and what role does it play in testing?
Coverage Principle

- The basic idea of coverage testing is that testing is complete when a well-defined set of tests is complete.
  - Example
    - Pilots use pre-flight check lists
    - Shoppers use grocery lists
to ensure the correct completion of their tasks

- In the same way testers can count the completed elements of a test plan
  - Example
    - Requirements
    - Functionalities
    - Blocks, Decisions (control-flow based)
    - C-uses, P-uses and All-Uses (dataflow-based)

The Role of Coverage in Testing

- We’re never done testing, and we don’t really know “how adequate” any test suite is
- Coverage testing: our best effort
  - Something sensible to measure
  - “Better than nothing” – usually much better
- It provides a way of monitoring and measuring the progress of testing against explicit quantitative completion criteria
  - Gives a clear measure of the completion of the testing task
Coverage is a Metric for Testing

<table>
<thead>
<tr>
<th>Source</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Number of requirements covered by tests</td>
</tr>
<tr>
<td>Requirements</td>
<td>Number of tests per requirement</td>
</tr>
<tr>
<td>Code</td>
<td>Proportion of lines of code (LoC) covered by tests</td>
</tr>
<tr>
<td>Code</td>
<td>Proportion of decisions covered by tests</td>
</tr>
<tr>
<td>Code</td>
<td>Proportion of conditions within decisions covered by tests</td>
</tr>
<tr>
<td>Code</td>
<td>Proportion of variable def-use relationships exercised by tests</td>
</tr>
<tr>
<td>Code</td>
<td>Proportion of loops that have been executed zero, one, and more than one times</td>
</tr>
<tr>
<td>Budget</td>
<td>How much of a project’s testing budget has been used?</td>
</tr>
</tbody>
</table>

*These metrics can be combined!*

What is Code Coverage Testing

- It is “White Box Testing”
- Takes into account the structure of the software being tested
- Measures how thoroughly the code has been tested with respect to certain metrics
  - Most basic, well-known, and widely used: Statement Coverage
  - “What lines were executed by the program during the tests?”
  - Basic insight: all code should be tested
- Strategy
  - Write tests that execute untested “parts” of your project
  - In theory, stop when you have achieved 100% coverage
    - In practice, should we? Why?
Code Coverage Testing versus Functional Testing

- When test inputs are generated using *program specifications*, we say that we are doing functional testing
  - Functional testing tests how well a program meets the *functional requirements*

- These two types of testing are complementary
  - Basic functionality should always be tested
  - The set of tests generated from functional testing provides a good basis for code coverage testing

- Functional testing: cover *requirements*
- Code coverage testing: cover *the code (implementation)*

History of Code Coverage Testing

- Using profiling tools to access the amount of code coverage during testing (1960’s)
- Using `tcov` to give statement coverage data for C and Fortran programs (1970’s)
- Two groups of test criteria
  - Controlflow-based testing (block & decision)
  - Dataflow-based testing (*c-use, p-use and all-uses*)
Importance of Code Coverage Testing

• In general, a piece of code must be executed before a fault in it can be exposed
  • Note: execution is necessary for fault detection, but it is certainly not sufficient. Why?
• Helps early fault detection
  – Are system testers finding faults that should have been found and fixed by developers?
  – Relative cost of fixing a software fault

State of Practice

• A published study (ICSE’92)
  – Coverage above 60-70% in system testing is very difficult
• Don Knuth’s system testing of TeX (23,000 LOC)
  – 85% block and 72% decision coverage (1992)
• Brian Kernigan’s testing of AWK
  – 70% block and 59% decision coverage (1991)
• Why is this so difficult?
Efficient Coverage Testing

• How much have I covered?
• What should I cover next?
• How should I do it?
• Software tools (partially) help with these issues.

Efficient Coverage Testing (1)

• How much of the code is currently tested?
  What is missing?
  – Which statements were exercised?
  – Which paths were traversed?
  – Which def-use associations were exercised?
  – Which functions got invoked from where?
• Need help in creating tests?
  – Which statement should I try to cover next?

We rely on advanced software tools for many types of coverage testing.

Tools can analyze the controlflow graph of the program to find the dominant blocks, decisions, and def-use pairs.
Testing Vocabulary: Basic Block

- A basic block is a sequence of consecutive statements or expressions, containing no branches except at the end, such that \textit{if one element of the sequence is executed all are.}

```
main() {
    c=0;
    i=0;
    while ( i<10 ) {
        c=c+1;
        i=i+1;
    }
    print(c);
}
```

A program, its control flowgraph, basic blocks, and decisions

---

Testing Vocabulary: Decision

- A decision is a boolean predicate with two possible values, \textit{true} and \textit{false}

```
if (fp) {  // example
    ...}
else if (condition1) { ...}
else if (condition2) { ...}
...
```

Simple versus compound decisions
Testing Vocabulary: Uses of Variables (C-use & P-use)

A Variable Use is a connection between:
   a definition (e.g. x = ...) and a (feasible) use
Mathematically, it is a pair (def, use)

Vocabulary: Uses of Variables (C-use & P-use)

main() {
    c=0;
    i=0;
    while ( i<10 ) {
        c=c+1;
        i=i+1;
    }
    print(c);
}

main() {
    c=0;
    i=0;
    while ( i<10 ) {
        c=c+1;
        i=i+1;
    }
    print(c);
}
Example of a Testing Tool

- Use prioritization and visualization to provide hot spots that give the most value in coverage.
- Each color represents a different weight determined by a control flow analysis using the concept of superblocks and dominators.

### Example of a Testing Tool

<table>
<thead>
<tr>
<th>File</th>
<th>Options</th>
<th>Summary</th>
<th>TestCases</th>
<th>Update</th>
<th>Debug</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.c</td>
<td></td>
<td></td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.c</td>
<td></td>
<td></td>
<td>80%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.c</td>
<td></td>
<td></td>
<td>60%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.c</td>
<td></td>
<td></td>
<td>40%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Code in white has already been covered by a test case and covering it again will not add new coverage.
- Covering this red block guarantees the execution of at least 8 additional blocks.
Example of a Testing Tool

Covering either true or false branch guarantees the execution of at least another 8 branches.

On to How These Tools Work

• Why should I care?
  – Using a software tool without understanding how it works is dangerous.

• My company won’t let me use tools like this.
  – You can think like a software tool and make your own manual testing more efficient.

• I don’t write tests, my testers do.
  – These concepts will help you to write more testable and maintainable – and thus better – software.
Dominator & Super Block (1)

- A super block consists of one or more basic blocks that if one block in the super block is executed all are
  - If any statement in a super block is executed, then all statements in it must be executed, provided the execution terminates on that input
  - A super block need not be contiguous

- Block u dominates block v if every path from entry to end, via v, contains u
  - u dominates v if covering v implies the coverage of u
  - Test execution cannot reach v without going through u

- Given a program, identify a subset of super blocks whose coverage implies that of all super blocks and, in turn, that of all basic blocks

- Why are superblocks so important?

Dominator & Super Block (2)

```c
#include <stdio.h>

int main() {
    int x = 0;
    while (x < 5) {
        if (x == 0) {
            break;
        }
        if (x == 3) {
            continue;
        }
        x++;
    }
    return 0;
}
```

An example C program

Control Flowgraph
Dominator & Super Block (3)

Quiz: Does node 4 or node 12 predominate node 13? Why?

Quiz: Does node 9 postdominate node 8? Why?
Dominator & Super Block (4)

- A strongly connected component of a basic block dominator graph has the property that every node in the component dominates all other nodes in that component.

Dominator & Super Block (5)

- Obtained by removing the composite edges in the right Figure on the previous slide.
- An edge $e$ from a node $u$ to a node $v$ is said to be a composite edge if $v$ is also reachable from $u$ without going through $e$.

- Only need to create test cases that cover basic blocks 4, 7, 9, and 10 - one from each leaf node in the super block dominator graph.
Back to the Code

Leaf Superblocks are circled: cover all four and the entire program is covered!

An example C program

Dominator & Super Block (6)

- At most four test cases need to be developed to cover all 14 basic blocks

The order in which the targeted basic blocks are covered and the corresponding cumulative coverages achieved.

- An alternative order is 10, 7, 4, and 9
Dominator & Super Block (7)

- Experimental results: does this work in practice?

<table>
<thead>
<tr>
<th>program</th>
<th>basic blocks</th>
<th>blocks that need to be covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>455</td>
<td>138</td>
</tr>
<tr>
<td>spiff</td>
<td>1266</td>
<td>361</td>
</tr>
<tr>
<td>mgr</td>
<td>3848</td>
<td>1043</td>
</tr>
<tr>
<td>ion</td>
<td>4886</td>
<td>1280</td>
</tr>
<tr>
<td>atac</td>
<td>8737</td>
<td>2574</td>
</tr>
<tr>
<td>odin</td>
<td>9870</td>
<td>2344</td>
</tr>
<tr>
<td>xlib</td>
<td>15580</td>
<td>5111</td>
</tr>
<tr>
<td>tvo</td>
<td>17680</td>
<td>6267</td>
</tr>
</tbody>
</table>

How xAtacs Works: Node Weighting

- The weight of a given node is the number of nodes that have not been covered but will be if that node is covered.

Example 1:
- To arrive at node 18 requires the execution also go through nodes 1, 2, 4, 7, 12 and 13.
- Node 18 is dominated by nodes 1, 2, 4, 7, 12 and 13.
- These nodes will be covered (if they haven’t been) by a test execution if that execution covers node 18.
- Assuming none of the nodes are covered so far, we say that node 18 has a weight of 7 because covering it will increase the coverage by at least 7 additional nodes.

- Why is it important to take a “conservative” approach?
  - Will node 6 be covered by covering node 18?
Example 2:
- Arriving at node 6 requires the execution only goes through nodes 1, and 2.
- Assuming none of the nodes is covered so far, we say that node 6 has a weight of 3.

Weight Re-computation (2)

The execution of certain tests may change the weights of nodes that are not covered by these tests.

After a test is executed to cover node 18, the weight (in terms of increasing the coverage) of node 6 is reduced from 3 to 1.