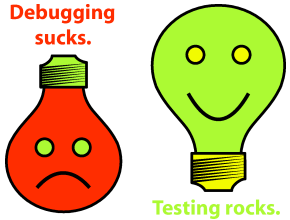

Testing on the Toilet
Understanding Your Coverage Data

Mar. 6, 2008

Code coverage (also called test coverage) measures which lines of source code have been executed by tests. A **common misunderstanding** of code coverage data is the belief that:

My source code has a high percentage of code coverage; therefore, my code is well-tested.

The above statement is **FALSE**! High coverage is a necessary, but not sufficient, condition.

Well-tested code ========> High coverage
Well-tested code <== X === High coverage

The most common type of coverage data collected is **statement coverage**. It is the least expensive type to collect, and the most intuitive. Statement coverage measures whether a particular line of code is ever reached by tests. Statement coverage does not measure the percentage of unique execution paths exercised.

Limitations of statement coverage:

- It does not take into account all the possible data inputs. Consider this code:

```
int a = b / c;
```

  This could be covered with `b = 18` and `c = 6`, but never tested where `c = 0`.

- Some tools do not provide fractional coverage. For instance, in the following code, when condition `a` is true, the code already has 100% coverage. Condition `b` is not evaluated.

```
if (a || b) {
  // do something
}
```

- Coverage analysis can only tell you how the code that exists has been exercised. It cannot tell you how code that *ought* to exist would have been executed. Consider the following:

```
error_code = FunctionCall();
// returns kFatalError, kRecoverableError, or kSuccess
if (error_code == kFatalError) {
  // handle fatal error, exit
} else {
  // assume call succeeded
}
```

  This code is only handling two out of the three possible return values (a bug!). It is missing code to do error recovery when `kRecoverableError` is returned. With tests that generate only the values `kFatalError` and `kSuccess`, you will see 100% coverage. The test case for `kRecoverableError` does not increase coverage, and appears "redundant" for coverage purposes, but it exposes the bug!

So the **correct way to do coverage analysis** is:

1. Make your tests as comprehensive as you can, without coverage in mind. This means writing as many test case as are necessary, not just the minimum set of test cases to achieve maximum coverage.
2. Check coverage results from your tests. Find code that's missed in your testing. Also look for unexpected coverage patterns, which usually indicate bugs.
3. Add additional test cases to address the missed cases you found in step 2.
4. Repeat step 2-3 until it's no longer cost effective. If it is too difficult to test some of the corner cases, you may want to consider refactoring to improve testability.

Reference for this episode: How to Misuse Code Coverage by Brian Marick from Testing Foundations.

More information, discussion, and archives:
http://googletesting.blogspot.com