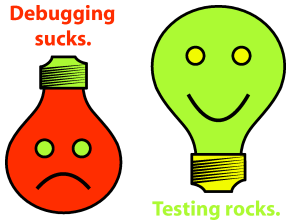

Testing on the Toilet

Too Many ~~Notes~~ Tests

Feb. 21, 2008

In the movie ***Amadeus***, the Austrian Emperor criticizes Mozart's music as having "too many notes." How many small tests are "too many" to test one function? Consider the method `decide`:

```
public void decide(int a, int b, int c, int d, int e, int f) {
  if (a > b || c > d || e > f) {
    DoOneThing();
  } else {
    DoAnother();
  }   // One-letter variable names are used here only because of limited space.
}     // You should use better names. Do as I say, not as I do. :-)
```

How many tests could we write? Exercising the full range of int values for each of the variables would require **2^{192} tests**. We'd have googols of tests if we did this all the time! **Too many tests.**

What is the fewest number of tests we could write, and still get every line executed? This would achieve 100% **line coverage,** which is the criterion most code-coverage tools measure**. Two tests**. One where `(a > b || c > d || e > f)` is true; one where it is false. **Not enough tests to detect most bugs** or unintentional changes in the code.

How many tests to test the logical expression and its sub-expressions? If you write a test of `decide` where `a == b`, you might find that the sub-expression `a > b` was incorrect and the code should have been `a >= b`. And it might make sense to also run tests where `a < b` and `a > b`. So that's three tests for `a` compared to `b`. For all of the parameters, that would **3 * 3 * 3 = 27 tests. That's probably too many.**

How many tests to test the logical expression and its sub-expressions independently? Consider another version of `decide`, where the logical sub-expressions have been extracted out:

```
public void decide(int a, int b, int c, int d, int e, int f) {
  if (tallerThan(a, b) || harderThan(c, d) || heavierThan(e, f)) {
    DoOneThing();
  } else {
    DoAnother();
  }
}
boolean tallerThan(int a, int b) { return a > b; }   // Note "package scope"
boolean harderThan(int c, int d) { return c > d; }   // rather than public; JUnit
boolean heavierThan(int e, int f) { return e > f; }  // tests can access these.
```

We can write four tests for `decide`. One where `tallerThan` is true. One where `harderThan` is true. One where `heavierThan` is true. And one where they are all false. We could test each of the extracted functions with two tests, so the total would be **4 + 2 * 3 = 10 tests**. This would be **just enough tests** so that most unintentional changes will trigger a test failure. Exposing the internals this way trades **decreased encapsulation** for **increased testability.** Limit the exposure by controlling scope appropriately, as we did in the Java code above.

How many tests is too many? The answer is "It depends." It depends on how much confidence the tests can provide in the face of changes made by others. Tests can detect whether a programmer changed some code in error, and can serve as examples and documentation. **Don't write redundant tests, and don't write too few tests.**

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