COSC345

Code Coverage

Richard A. O'Keefe

Outline

- Why do we care about coverage?
- What is it?
- How do we measure it?
- What do we learn from it?
- Resources

I care about coverage because

- My programs don't work
- Your programs don't work
- Nobody's programs work
- At least not all the time.

How to stop broken code at the door

- Static checking. gcc -O2 -Wall -Wextra is pretty much minimal for C. clang –analyze is good. lint (closed) and splint (open) are useful.
- Formal verification is *wonderful* but hard.
  seL4 “The world's first operating-system kernel with an end-to-end proof of implementation correctness and security enforcement”.
  - 8,700 lines of C code plus 600 lines of ARM assembly code.
  - 7,500 lines of C code verified.
  - 16 bugs found by testing, **144** by verification
  - took 25 work-years

How to stop broken code 2

- Inspection
  - Formal inspection follows a specific process with teams looking at the code.
  - Informal inspection, also called desk checking.
    It's amazing what you can see by looking.
  - Inspection is said to be very effective at finding bugs
- Testing
  - Put system into a known state, feed it input where you know something about what it should do, see if it does that.

If it's not tested it doesn't work

- Compilers do not find all errors.
- Inspection does not find all errors.
- Testing does not find all errors.
- All of them together do not find all errors, but they get quite far.
- If you haven't tested a piece of code, you have no right to believe it correct.
- THAT is why we care about coverage.

A taste of what's coming

- I have an example for this lecture, the lexical analyser of a compiler for a language called Pop-2.
- It has been tested on 10s of 1000s of lines of source code.
- Test coverage immediately showed lots of code had never been tested.
- Oops, I was testing it with GOOD source code, so the error reporting code didn't get used!

Error reporting? Is that important?

- Error reporting and handling code is often difficult to test.
- Suppose you run a hospital. It needs an emergency power generator. What do you think will happen if mains power fails and you have not tested your generator regularly?
- Error handling must not make things worse.
- try {…} catch (…) {… /*FALLTHROUGH*/}
  is stunningly common in Java code oops

What is coverage?

- Simply, determining and reporting which parts of your code have been used in testing and which have *not*.
- **Function** coverage: has every function in your program been called at least once?
- **Entry/exit** coverage: has every call to every function been executed at least once? has every possible return from that function been executed at least once?

What is coverage? 2

- **Line** coverage: has every line of your code been executed at least once?

  Note: if (p()) q(); else r(); is one line.
- **Statement** coverage: has every statement of your code been executed at least once? You should aim for this as the *minimal* level of coverage you want to achieve.
- **Branch** coverage: has every alternative of **if** and **switch** statements been executed?

What is coverage? 3

- **Condition** coverage: has every *condition* (boolean function, comparison, &c) been true and been false at least once each?
- if (a() && b()) q(); else r();
  needs a() false, a() true b() false, a() true b() true.
- We normally want branch+condition coverage.

Coverage based on values

- float f(float x) { return sqrtf(x – 1.0f); }
- Calling that f(2.0f) gets function, entry/exit, line, statement, branch, and condition coverage.
- But that doesn't mean f(0.5f) will work!
- **Parameter** coverage asks whether all semantically different combinations of values are tested (or impossible).
- -∞, -large, -small, -0, +0, +small, +large, +∞ and NaN

Boundary Testing

- Parameter value coverage is related to boundary testing, where we start by dividing parameter space into regions that should be handled the same way and construct tests to drive evaluation into each of those regions.
- Strings: null, empty, white space, single character, extremely long, valid data syntax, invalid data syntax, …
- float f(float x) { **assert**(x >= 0); return sqrtf(x – 1.0f); } turns this example into condition coverage.

How do we measure coverage?

- Build our program with a coverage tool.
  - This *instruments* the program by adding code to count how often control reached each basic block. Concurrent programs must use atomic adds.
- Run tests.
  - The instrumented program dumps the counts in files, these counts accumulate so many runs Is ok
- Use the companion reporting tool.
  - Combines source information with counts to produce some kind of report.

What do we learn? Example

- www.cs.otago.ac.nz/cosc345/coverage.txt
- gcc stores fixed coverage-related information derived from the source code in *.gcno files
- running the program stores dynamic counts in non-text *.gcda files. We need to clear old ones out.
- We have to compile the program with the right options. With some compilers it's one option; with gcc it's two, and you need both.
- Nothing special about running the program here. Python might use a special command

What we learn, example, 2

- gcov -f prints a summary about functions
- OOPS! I have not achieved function coverage. The bracket_error function is not called.
- New test case: echo “)” | a.out
- OOPS: it gave an error message but the wrong one, bracket_error still wasn't called.
- New test case: echo “]” | a.out
- That worked: bracket_error *was* called, and now I know what was wrong the first time.
- There **was** a bug that had been there a year because there was no test case for it.

What we learn, example 3

- Look for #### in *.gcov files.
- enable_keyword(): certain optional keywords were not enabled for these test files, so they are not routinely tested. (I do have tests for them.)
- integer literals too big for 64 bits are not tested
- illegal radices are not tested
- operator priorities > 9 are not tested
- ill-formed floating point literals are not tested
- 1.2e3 is 1 token, 1.2e is 2, the latter not tested
- over-large floating point literals are not tested

Resources

- clang and gcc documentation
- gcov(1) documentation
- Jcov for java “used with JDK from the very beginning of Java … open source … the only code coverage tool working with a JDK release in development”
- Python has Coverage.py, figleaf, others?
- Haskell has a great tool, example 2 if time.