COSC345 Week 7

Test-Driven Development

11 April 2017

Richard A. O'Keefe

Meltdown in Alberta

- The University of Alberta has a collection of ice cores
- They are stored at -37°C, worked on at -25°C.
- The cores were moved in on March 24.
- On April 2, 12.8% of the collection melted.
- Freezer failure $\Rightarrow$ heat pumped into cold rooms
- "The system monitoring the freezer temperature failed due to a database corruption. The freezer's computer system was actually sending out alarm signals that the temperature was rising, but those signals never made it to the university's service provider or the on-campus control centre."

Issues

- “a database” was corrupted (bug 1)
- the corruption was not detected (bug 2)
- the corruption was able to block alarms (bug 3)
- the high heat alarm went off too late (bug 4)

$\oplus$ 2 years to prepare, failed on 10th day of operation

If it isn't tested, it doesn't work.

- A slogan from Agile Programming

- If it isn't tested, you have no right to *believe that or act as if* it works

- Matches my experience: code that *looks* right often *is* wrong, especially in dynamic languages.

Example: UTF16-LE

```
function encode(s, n) {
    if (n < 0x10000) {
        putbyte(s, n&255); putbyte(s, n >> 8);
    } else {
        encode(s, 0xD800 | ((n - 0x1000) >> 10));
        encode(s, 0xDC00 | (n & 0x03FF));
    }
}

// See the mistake?
```

Let's see it again

```
function encode(s, n) {
    if (n < 0x10000) {
        putbyte(s, n&255); putbyte(s, n >> 8);
    } else {
        encode(s, 0xD800 | ((n - 0x1000) >> 10));
        encode(s, 0xDC00 | (n & 0x03FF));
    }
}

// Should be the same number!
```

How do you find that?

- Get a person to find it.

— We'll talk about formal inspection later.

- Get a program to find it.

— If a function contains two magic numbers that are related by a single edit, ask if they should be the same number.

- TEST!

Finding it 2

• I have a program that gets part of the way. I've found bugs in other code using it. Made a mistake once? You probably made it again.

• The broken code had been there for about three years. It has been looked at a *lot*. I recently found a different error in related code, where 0xDC00 and 0xD800 had been swapped. People aren't good with numbers.

Finding it 3

- Write a test program.
- Encode all the Unicode characters.
- See if they read back correctly.
- Wrote 65536 but read 55356.
- Test failed with a clue about why.

What was found

- The error you've already seen (twice).

- The base 0x10000 was sometimes restored using | instead of + so 131072 read as 65536 (twice).

- 1 error every 17 SLOC of the code under test (not locatable by the compiler)..

- Test took 19 SLOC (25% of source) but paid off. Those errors are not coming back!

What I did wrong I

- I didn't write the test first.
- If I had, I would have *run* the test as soon as the code existed.
- The mistakes would have been found three years sooner.
- **While I still remembered how it was supposed to work.**

Test-Driven Development

- Before you make a change:

— Write at least one test case for it

— which must fail.

- When you have made the change:

— Run the test case(s).

— Debug if necessary.

TDD 2

• When you're done, maybe refactor the code to keep the structure clean.

• *Whenever you make a change*,

— **Run all the tests**.

Dialectical Mistakialism

• A slogan going back to Hegel is "the transformation of quantity into quality".

• A fancy way of saying "enough more and it's not just more, it's *different*".

• Pull one hair from your head, who can tell? Keep it up and you're bald. That's a difference.

Quantity to Quality

- There is no universal *law* about quantity transforming to quality,

- but it's common enough to watch out for.

— Increase in memory used $\rightarrow$ thrashing.

- And to watch for a chance to exploit it.

— Gigatown Dunedin? Phones? Smartwatches? 3D printing? Deep learning? Data mining?

And testing?

• Old days: machine slow, expensive, shared by many people; get 2 or 3 runs per day.

— Therefore do lots of "desk checking", plan testing very carefully, test only what you need to.

• These days: machines fast, cheap, unshared; get nearly instant turnaround.

Frequently run all tests (Travis)

- For large systems, general rule is to run all tests overnight.

— Check the program builds on all systems.

— Unit tests.

— System tests.

— Performance tests, leak tests, ...

Fixing a component?

• Run at least all the tests for that component before putting it in the shared repository.

• After all, you have enough trouble with mistakes in your own code; you don't want to have to worry about someone else's!

Most bugs are shallow

- Producing test cases doesn't have to be hard.

- Any test case is better than none.

- x = **new** Set(); **assert** x.isEmpty();
x.add(137); **assert** x.size() == 1;
x.remove(136); **assert** x.size() == 1;
x.remove(137); **assert** x.size() == 0;

In future lectures:

- The assert statement is your friend
- Unit testing frameworks
- JUnit and relatives.
- QuickCheck, PropEr, and relatives.