Lecture 17 — More Testing

Jeff Zarnett & Patrick Lam
jzarnett@uwaterloo.ca & p.lam@ece.uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

December 29, 2014

Regression Testing

The name “regression” testing comes from the desire that software goes forward (“progress”) not backwards (“regress”).

Previously fixed bugs should stay fixed.

Regression Testing

“any software testing that uncovers errors by retesting the modified program” (Wikipedia).

Often refers to large suites of test cases which detect regressions:

- of a bug fix that a developer has proposed.
- of related and unrelated other features that have been added.

Regression Testing: Attributes

- **Automated**: no reason to have manual regression tests.
- **Appropriately Sized**: suites that are too-small will miss bugs; suites that are too-large take too long to run.
- **Up-to-date**: ensure that tests are valid for the version of program being tested.

Stress Testing

Stress testing is common in industry where reliability matters.

Testing the system “under stress”.

Find out how it performs “under pressure”.

Systems often crash, hang, or behave badly when under the right (wrong) kind of pressure.

Stress Testing

Programs under stress are *constrained in some resource*.

Adding more concurrent users is one way.

Some systems start having trouble at 2 concurrent users.
Others can easily handle thousands.

Stress Testing

Other things to restrict: CPU, RAM, disk space, network speed.

Surprisingly many programs crash when the hard disk is full!

Also might test unreliable network, bad RAM, etc.

Stress Testing: Inevitability

There will be some point where the system will fail.
- No system is invincible.

The only hope: fail gracefully.

Example: Tell the user he can’t log in: server full.
- The user is mad, but it’s better than crashing the server.
- The people logged in can continue, at least.

Stress Testing: Simulation

Stress testing is usually just a simulation.

Test setup of 50 computers $\times$ 1000 clients + user simulators.
A good stress test, but not identical to 50 000 users.

Why not?

System Testing

System testing is not an area of focus for this course.

Test the system by using it as users will use it.

Acceptance Testing.

Tests in Practice

Practices used in industry:

- **Unit Tests:** Modify the class, modify the test.
- **Code Reviews:** Branches/modules have owners.

Tests in Practice

Practices used in industry:

- **Continuous Builds:** A machine continuously checks out and tests the latest code.
  All unit and regression tests are run; status made public.
  Social pressure: if you break something, everyone knows!
- **QA Team:** If all tests have passed, a QA team will look for additional bugs.
- **Release:** Once QA has approved a build, it is released for use.

Bug Tracking Systems

If you don't write it down, you'll forget it.

Defect tracking systems keep lists of defects in a database (these days, often web-accessible).

Tracking systems keep track of:

- reported defects and their confirmations;
- who is assigned to fix the defect; and
- defect status.

Close a defect by changing its status to "resolved".

Popular web-based example: Bugzilla[1].

[1] `http://www.bugzilla.org`

Choosing Test Inputs

We talked about creating tests, but we haven't covered choosing inputs to them.

Inputs matter, of course, because the test need input to run.

How can you test `int add(int number1, int number2)` without choosing values for `number1` and `number2`?

Why Choose?

Why choose? Let’s test all possible inputs!

Assumptions

Let us assume we have:

A 32-bit computer

This means an integer has 2^{32} possible values.

A computer that can run 1 billion (1×10^9) tests per second.

1-Parameter Function

```
public int compute(int number1)
```

How long will it take to test this exhaustively?

1-Parameter Function

```
public int compute(int number1)
```

How long will it take to test this exhaustively? 0.07 minutes.

1-Parameter Function

Wait a minute!

You said this was impossible!

2-Parameter Function

```
public int compute(int number1, int number2)
```

How long will it take to test this exhaustively? 584.9

2-Parameter Function

```
public int compute(int number1, int number2)
```

How long will it take to test this exhaustively? 584.9 YEARS!

584.9 Years

This test, if started in the year 1428 would finish about now.

Christopher Columbus sailed for the new world in 1492.
In 1428 he had not even been born.

Sanity Check

Are our numbers correct?

The first 32-bit Processor (Motorola 68k) came out in 1979.

It did about 700 000 instructions per second.

Assume the function is so simple it is executed in 1 operation.

It would take the 68k about: 835 362 years to run.

If we started this test in 1979, in 2013 we'd be about: 0.00407% done.

3-Parameter Function

```
public int compute(int number1, int number2, int
number3)
```

How long will it take to test this exhaustively?

3-Parameter Function

```
public int compute(int number1, int number2, int
number3)
```

How long will it take to test this exhaustively?
2.512 trillion years
or... about 180 times the age of the universe.

Choosing Inputs

Conclusion: We have to choose our inputs (carefully).

Valid and Invalid Inputs

Typically we choose “good” (valid) inputs.

It’s also important to choose invalid inputs.
(Note: this makes the exhaustion testing problem worse!)

What does the spec say it should do?

Example: Fibonacci function given -1 as input.

Black Box and White Box Testing

Two ways we might write tests:

Black-Box Testing

and

White-Box Testing

Black Box Testing

Tests written without looking at the source code.

The code being tested is treated as a “Black Box”.

Sometimes someone else writes them.

Test-Driven Development: Write tests 1st; can’t look at a non-existent implementation.

White Box Box Testing

Gets its name because it’s the opposite of Black Box;
we look at the source.

Analyze the source code and use results to write tests.

Examine if-statements; make the code take all branches.

White Box Box Testing

Look for magic numbers (values).

If you see a statement `if (variable > 42)`.

We know 3 possible test inputs for `variable`:

1. A number > 42;
2. A number < 42; and
3. 42 (the edge case)

Final Note on Testing

Testing is potentially endless: no way to say we have found all the bugs.

At some point, we have to ship it.

Testing is a tradeoff between budget, time, and quality.

Use engineering judgement to decide how much is “enough”.

For more, take the Software Testing course!

Final Note on Testing

Changing the Culture of Testing at Google:
`https://www.youtube.com/watch?v=q6_xNC8NA2g`

Reason Number 501 Not to Deploy Buggy Software:
`https://www.youtube.com/watch?v=O8KJC_U3KjM`