Software Testing, Quality Assurance & Maintenance—Lecture 1

Patrick Lam
University of Waterloo

January 4, 2010

"The Testing Course"

Course mechanics

Textbook: I will be (somewhat loosely) following Ammann and Offutt.

Website: http://patricklam.ca/stqam

Grace days: You may submit assignments up to 4 days late in total.

Why Test?
Source: Wikipedia

Let's consider:

- consequences;
- causes;
- avoidance (before it's too late);
  - testing
- mitigation (afterwards).

Consequences of Failures

Who suffers from failures?

Photos: (L) epicfail.com; (R) copyright ESA/CNES/ARIANESPACE - Service Optique CSG

Consequences of Failures

Microsoft®

http://hermosodia.wordpress.com/2008/10/19/definicion-visual-de-workaround/

(United States Centre for Disease Control, 04MI074)

(stephen mantler at Flickr, "A runner's injury")

1. Segfaults—or crashes; infinite loops too.

Why Does Software Go Wrong?

```
public int add(int x, int y) {
    return x - y;
}
```

2. Wrong Output:
   - method or module returns wrong information or has unwanted side effect.

Why Does Software Go Wrong?

3. Wrong API
- a library can't do what you need it to do; or
- subsystems don't work together correctly.

Photo copyright ESA/CNES/ARIANESPACE - Service Optique CSG

Why Does Software Go Wrong?

4. Bad system-level behaviour:
   - Wrong output to user.
   - Bad security.
   - Wrong specifications.

Why Does Software Go Wrong?

5. Nonfunctional properties:
   - Leaks.
   - Performance.

Regressions to past bugs.

Avoiding Software Failures

- test the software (in-house, externally)
- require validation suites for plugins
- code review
- better design ("write better code!")
- include fewer features
- defensive programming (especially wrt plugins)

Software never completely works.

Aim: make software that is good enough.

Coping with an Imperfect World

- disclaim liability

  25. LIMITATION ON AND EXCLUSION OF DAMAGES. You can recover from Microsoft and its suppliers only direct damages up to the amount you paid for the software. You cannot recover any other damages, including consequential, lost profits, special, indirect or incidental damages.

  (Vista license)

Coping with an Imperfect World

- disclaim liability
- release patches
- backup user data
- defensive programming

Ways of Testing Software

- compile it

- compile it
- run it on one input

Ways of Testing Software

- compile it
- run it on one input
- run it on many inputs

Ways of Testing Software

- compile it
- run it on one input
- run it on many inputs
- run it on a representative set of inputs

- compile it
- run it on one input
- run it on many inputs
- run it on a representative set of inputs
- run it on all inputs (static analysis)

Other Testing Concerns

- Integration testing
- Nonfunctional properties
- Regression tests

Goals of This Course

- You will be able to create and evaluate test suites for reasonably-sized software systems.
- You will be exposed to tools for software maintenance and verification.
- You will gain experience with carrying out modifications to a large pre-existing software package.
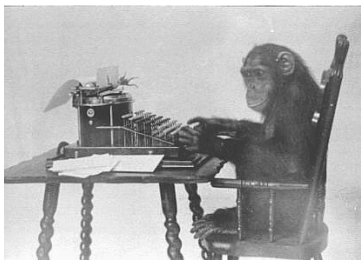
You'll find bugs eventually, but it'll take longer than you have.

Idea: find a reduced space and cover it with tests.

Idea: find a reduced space and cover it with tests.

Possible spaces: graphs, logic, input space, syntax.

Concrete Example of Coverage

```
public static numZero(int[] x) {
  int count = 0;
  for (int i = 1; i < x.length; i++) {
    if (x[i] == 0) count++;
  }
  return count;
}
```

Input space: arrays of length ≤ 2 containing 0 or 1.

$$[], [0], [1], [0, 0], [0, 1], [1, 0], [1, 1]$$

Note: Only some of the inputs will trigger the failure.

Validation: evaluating software prior to release to ensure compliance with intended usage.

Verification: determining whether products of a given phase of the development process fulfill requirements established in a previous phase.

Software fault: static defect in the software.
Software error: incorrect internal state that is the manifestation of some fault.
Software failure: External, incorrect behaviour.

Testing: evaluating software by observing its execution.
Debugging: finding (and fixing) a fault given a failure.

- Testing tasks are often repetitive (i.e. boring).
- It is easy to make mistakes while carrying out tests.

Automation is key to successful testing:

- Enables you to run more tests more quickly.
- Helps ensure coverage.

Tools

We'll discuss a number of software tools, probably in the tutorials.

- version control (e.g. CVS, Subversion, git)
- build management (make, ant, etc)
- tinderboxes
- bug databases

More tools

I also plan to talk about some more verification-specific tools:

- JUnit
- JML tools
- ESC/Java2
- Software model checking

A Collection of Topics

Standard topics:

- Graph Coverage
- Logic Coverage
- Input Space Partitioning
- Syntax-Based Testing (including mutation testing)
- Testing in Practice (e.g. test plans, writing bug reports)

Non-standard topics:

- concurrency
- failure-obliviousness
- exception handling
- testing for performance