

Software Testing, Quality Assurance & Maintenance—Lecture 1

Patrick Lam
University of Waterloo

January 5, 2026

A problem has been detected and windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

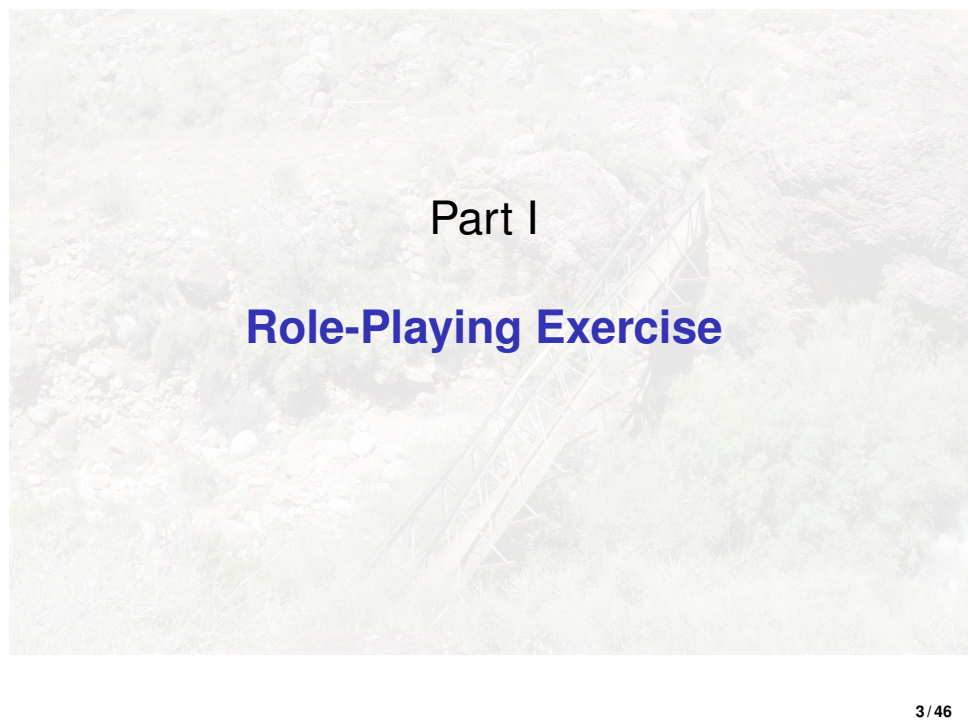
Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

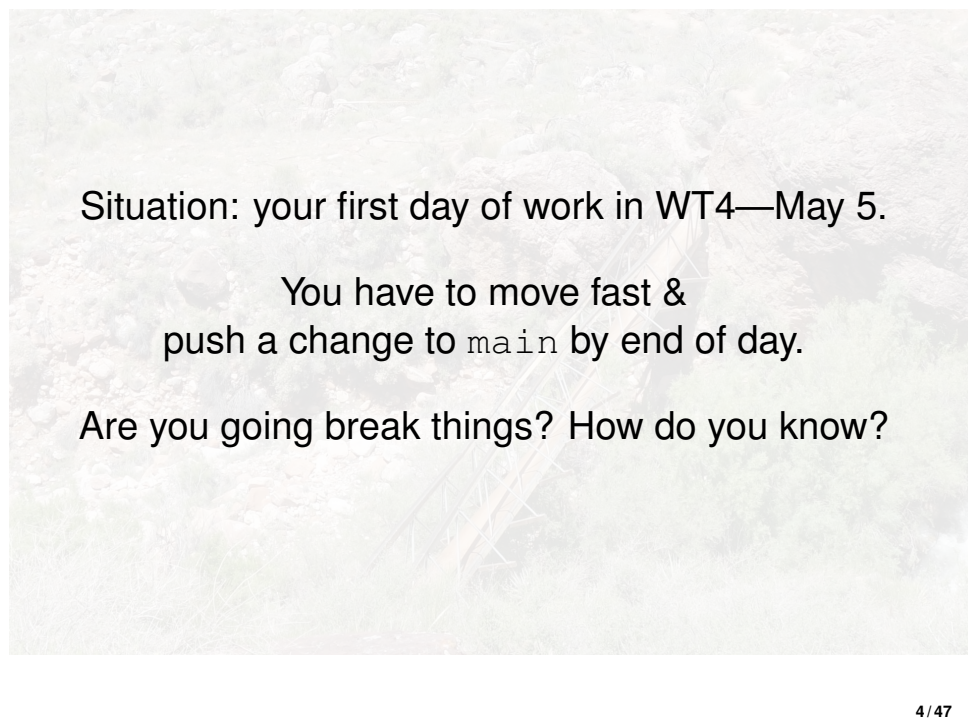
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

The background of the slide is a faded, high-angle photograph of a rugged, rocky hillside. A metal structure, possibly a staircase or a walkway, is visible on the right side of the image, winding up the slope. The terrain is covered with sparse vegetation and numerous rocks of various sizes.

Part I

Role-Playing Exercise



Situation: your first day of work in WT4—May 5.

You have to move fast &
push a change to `main` by end of day.

Are you going break things? How do you know?



Details

You are working for:

- mom & pop website design shop?
- a tech giant?
- Tesla?

Are you going break things? How do you know?
What is the consequence?

Avoiding Software Failures

Consider this spectrum:

- YOLO
- ad-hoc testing (manual tests)
- **ad-hoc testing** (automated tests)
- **principled testing** (tools)
- linting / type systems
- **formal verification**

Techniques to Avoid 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 for plugins)

Thesis

The thesis of this course is that engineers must choose the right tools to make their code fit-for-purpose.

Learning Outcomes

- write good test suites;
- use tools to improve software quality
- prove software correct using tools
(beyond SE 212)

Which tools and techniques?

- coverage
- fuzzing
- sanitizers
- mutation-based analysis
- metamorphic testing
- Dafny, Kani, bounded model checking

Part II

Failures, including software failures

Failures

Let's consider:

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

Some Failures



Who suffers from failures?

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

More Failures



<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")

Infamous Software Bugs

Crowdstrike, 2024

Therac-25, 1985–1987:

5 deaths, severe injuries

race conditions, no automated testing

Northeast blackout, 2003

(no ice storm)

Ariane 5 crash, 1996

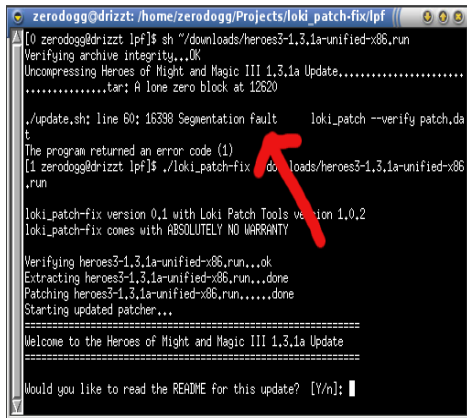
Morris Worm, 1988

Why Does Software Go Wrong (discussion)?

Why Does Software Go Wrong?

- ❶ crashes and infinite loops;
- ❷ wrong output;
- ❸ wrong API;
- ❹ bad system-level behaviour;
- ❺ nonfunctional properties;
- ❻ regressions.

Why Does Software Go Wrong?



A terminal window titled 'zerodogg@drizzt: /home/zerodogg/Projects/loki_patch-fix/lpf' showing a command sequence. The user runs a script to update 'Heroes of Might and Magic III 1.3.1a'. The script verifies the archive, then attempts to extract and patch it. A red arrow points to the error message: 'Segmentation Fault' on line 60 of the update script. The error occurs while running 'loki_patch --verify patch.dat'. The terminal output continues with version information and a welcome message for the update.

```
zerodogg@drizzt: /home/zerodogg/Projects/loki_patch-fix/lpf
[0 zerodogg@drizzt lpf]$ sh ~/downloads/heroes3-1.3.1a-unified-x86.run
Verifying archive integrity...OK
Uncompressing Heroes of Might and Magic III 1.3.1a Update.....
.....tar: A lone zero block at 12620

./update.sh: line 60: 16398 Segmentation Fault      loki_patch --verify patch.dat
The program returned an error code (1)
[1 zerodogg@drizzt lpf]$ ./loki_patch-fix ~/downloads/heroes3-1.3.1a-unified-x86.run

loki_patch-fix version 0.1 with Loki Patch Tools version 1.0.2
loki_patch-fix comes with ABSOLUTELY NO WARRANTY

Verifying heroes3-1.3.1a-unified-x86.run...ok
Extracting heroes3-1.3.1a-unified-x86.run...done
Patching heroes3-1.3.1a-unified-x86.run.....done
Starting updated patcher...

=====
Welcome to the Heroes of Might and Magic III 1.3.1a Update
=====

Would you like to read the README for this update? [Y/n]:
```

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.

```
chus@ATAHUALPA:~$ ./xxx
-----
Linux vmsplICE Local Root Exploit
By qaaz
-----
[+] mmap: 0x0 .. 0x1000
[+] page: 0x0
[+] page: 0x20
[+] mmap: 0x4000 .. 0x5000
[+] page: 0x4000
[+] page: 0x4020
[+] mmap: 0x1000 .. 0x2000
[+] page: 0x1000
[+] mmap: 0xb7d72000 .. 0xb7da4000
[+] root
root@ATAHUALPA:~# id
uid=0(root) gid=0(root) grupos=20(dialout),24(cdrom),25(floppy),29(audio),
,44(video),46(plugdev),106(netdev),109(powerdev),1000(chus)
root@ATAHUALPA:~#
```

Why Does Software Go Wrong?

5. Nonfunctional properties:
 - Leaks (yes, even in Java).
 - Performance.

Why Does Software Go Wrong?

6. Regressions to past bugs.

Mitigation: Failure is Inevitable

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/replicate user data
- defensive programming



Part III

Course Logistics

Course mechanics



Textbook: none

Github TBA

Piazza (you know where to find it)

Grace days: You may submit assignments up to 3 days late in total.

Course staff



Instructor:

Patrick Lam

TA:

Aosen Xiong

Evaluation

| | |
|--|-----|
| 3 individual assignments | 36% |
| 2 quizzes (1 hour, class time, STC 0040) | 20% |
| Final exam | 44% |

Quizzes, final are open-book, open-notes, no Internet.

Part IV

About This Course

Goals of This Course

- You will be able to create and evaluate test suites for reasonably-sized software systems.
- You will learn how to use and write tools for software maintenance and verification (particularly automated testing tools).

Thesis

The thesis of this course is that engineers must choose the right tools to make their code fit-for-purpose.

Learning Outcomes

- write good test suites;
- use tools to improve software quality
- prove software correct using tools (beyond SE 212)

Part I: Test Suites and Fuzzing

- writing unit tests;
- when to stop writing tests (coverage, mutation analysis);
- automatically writing tests with fuzzing (also greybox and with grammars)

Part II: Finding Key Inputs via Symbolic Execution

- symbolic execution;
- program semantics;
- dynamic symbolic execution.

Part III: Proving Programs Correct

- Dafny;
- Bounded model checking: CBMC and Kani;
- real-life applications.

Part IV: Grab-bag of Leftovers

- Concurrency
- Undefined Behaviour
- Metamorphic testing
- Security
- Library upgrades
- Continuous integration, code review, bug reports

Part V

Introduction to Testing

`www.fuzzingbook.org/html/Intro_Testing`

Summary

- introduced a `my_sqrt()` function
- manually tested it
- created testing infrastructure `assertEquals`
- generated tests for it
- added input validation
- saw the limits of testing with `my_sqrt(0)`

Part VI

Defining some terms

Terminology

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.

Terminology

Software fault: static defect in the software.

Software error: incorrect internal state that is the manifestation of some fault.

Software failure: External, incorrect behaviour (as in “epic fail”).

RIP model

Faults become failures by:

- being **R**eachable;
- **I**nfecting the program state; and
- **P**ropagating to the output.

Testing vs. debugging

Testing:

evaluating software by observing its execution.

Debugging:

finding (and fixing) a fault given a failure.

Bonus: Debugging and the Scientific Method

Don't: randomly debug your code.

Do: Make hypotheses and verify them.

Reference: Andreas Zeller. *Why Programs Fail: a Guide to Systematic Debugging*.