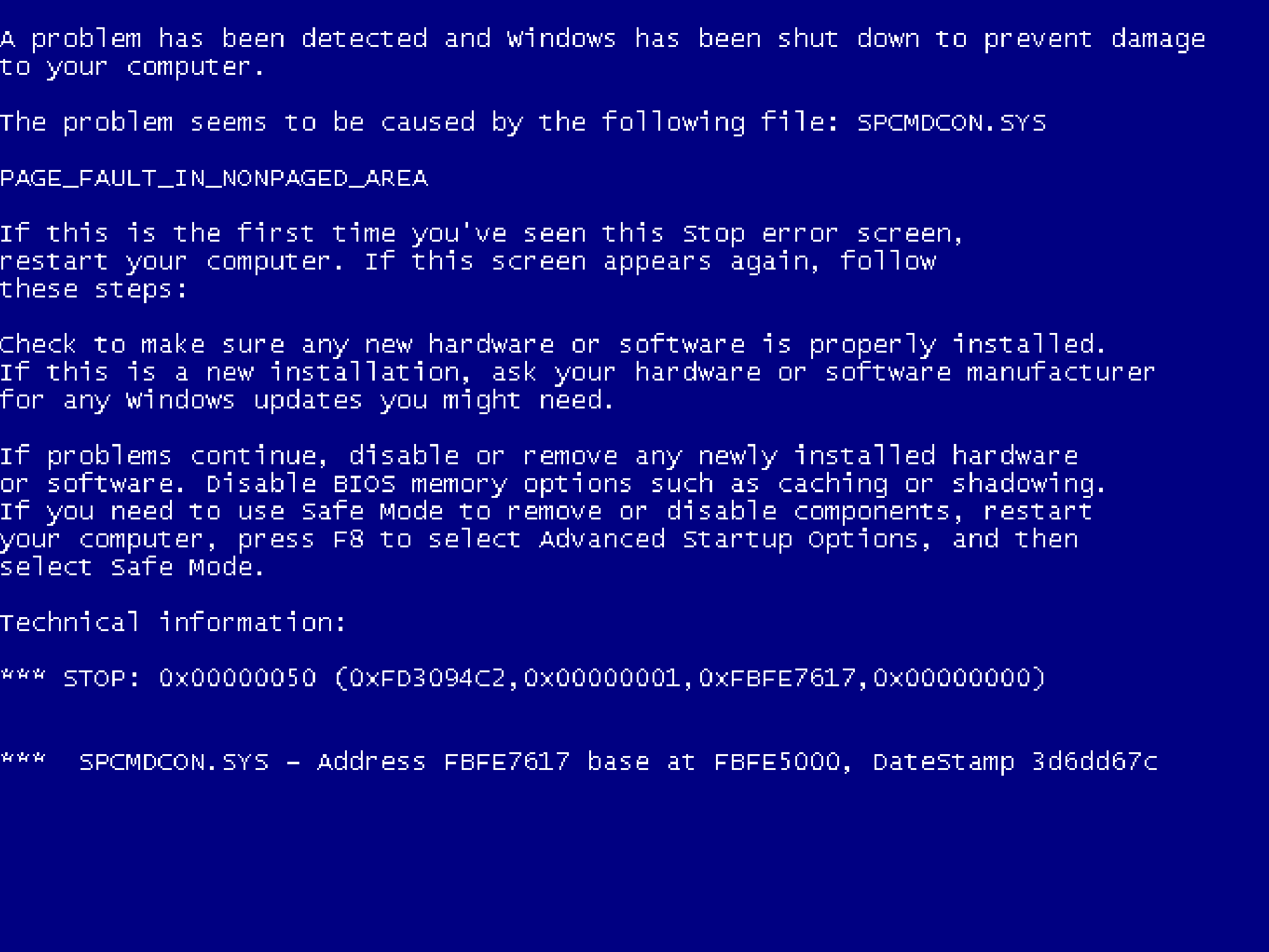
# Software Testing and Quality Assurance—Lecture 1

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

# Part I

# Role-Playing Exercise

Situation: your first day of work
on your next work term—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
- **bounded model checking**
- **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 uality
- prove software correct using tools
  (beyond SE 212)

# Which tools and techniques?

- coverage
- fuzzing
- sanitizers
- mutation-based analysis
- metamorphic testing
- bounded model checking (CBMC, Kani)
- formal verification (Dafny)

# Part II

# Failures, including software failures

# Failures

Let's consider:

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

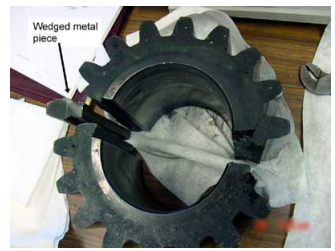
# Some Failures


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

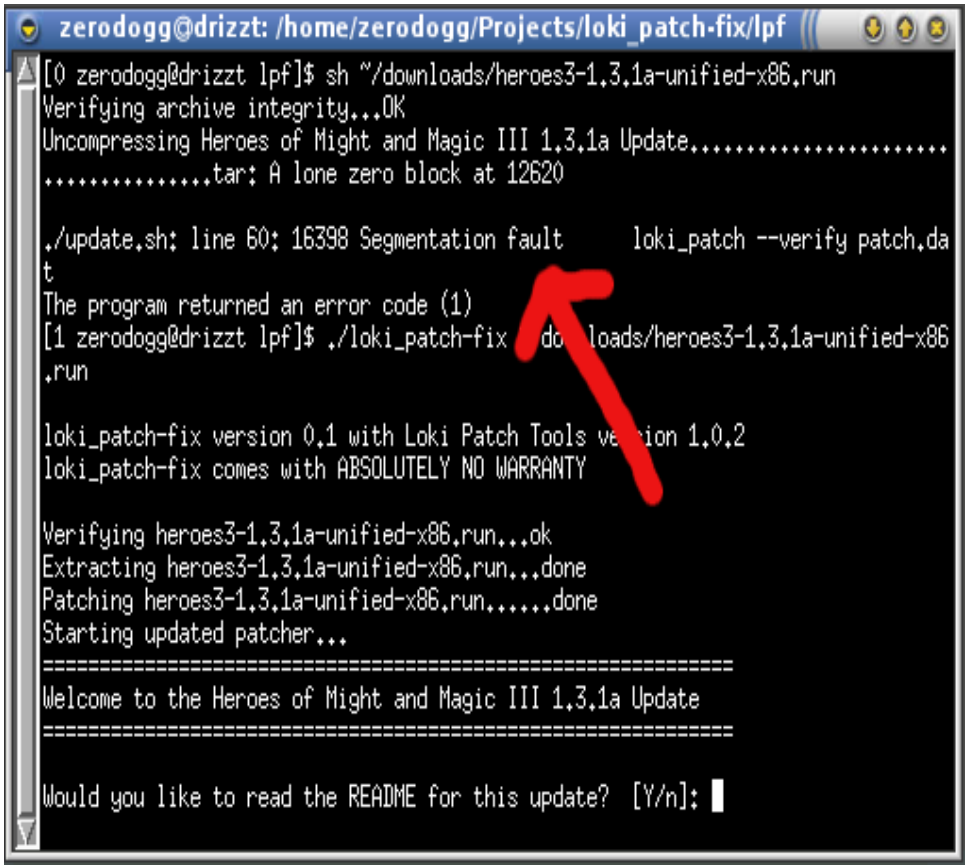
Morris Worm, 1988

# Why Does Software Go Wrong (discussion)?

# Why Does Software Go Wrong?

1. crashes and infinite loops;
2. wrong output;
3. wrong API;
4. bad system-level behaviour;
5. nonfunctional properties;
6. regressions.

# Why Does Software Go Wrong?




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.

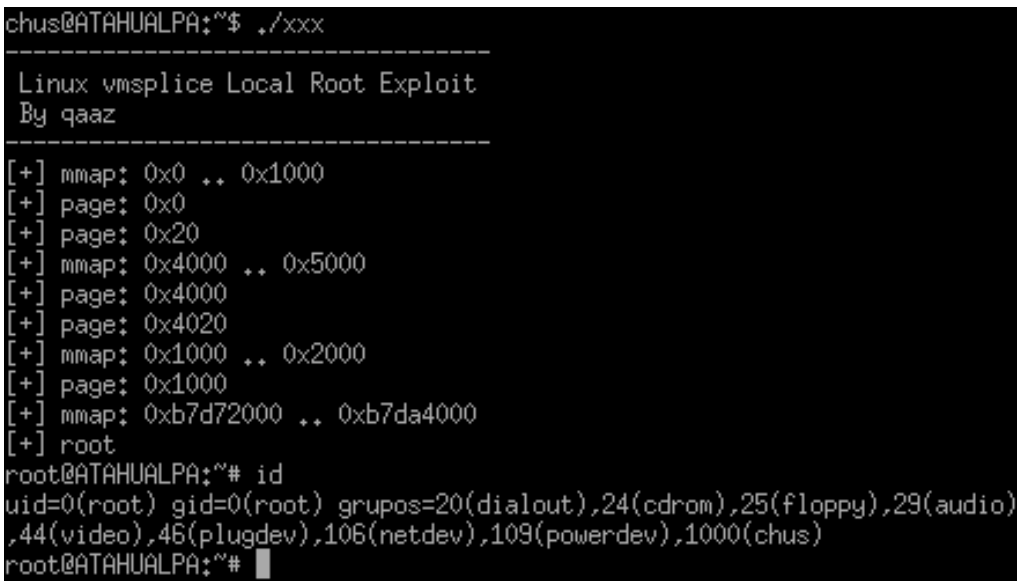
# Why Does Software Go Wrong?

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 (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

Gitlab `https://git.uwaterloo.ca/stqam-1261/pdfs`

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 |
| TAs: | Billy Bai |
| | Alex Le Blanc |

# Evaluation

| | |
|---|---|
| 3 individual assignments | 30% |
| 1 midterm (February 23) | 20% |
| Final exam | 50% |

Midterm and final are open-book, open-notes, no Internet.

# Part IV

# About This Course

# Goals of This Course

You will be able to:

- write new test suites, improve existing test suites, and critique test suite quality using engineering judgment
- use and develop tools (e.g. fuzzers, sanitizers, symbolic executors) to improve software quality
- use tools for automatic program verification (e.g. Dafny) or bounded model checking (e.g. CBMC/Kani)

# Thesis

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

# Module I: Engineering Test Suites

- writing unit tests;
- when to stop writing tests (coverage, mutation analysis);
- oracles;
- assorted other techniques, e.g. metamorphic testing, etc.

# Module II: Fuzzing

- automatically writing tests with fuzzing;
- mutation-based fuzzing;
- grammar-based fuzzing;
- fuzzing inputs for APIs;
- constraint-based fuzzing / property-based testing / automatic test generation;
- reducing failure-inducing inputs.

# Module III: Bounded Model Checking

- symbolic execution;
- bounded model checking for C and Rust;
- loop unwinding, proof harnesses, contracts

# Module IV: Proving Programs Correct

- Dafny;
- real-life applications thereof

# 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.*