Software Testing, and Test Case Design

CMPT 145

Software errors

- Easy to make.
- Will reduce the value of the software.
- Can be harmful to people.
- Can cause damage to equipment/environment.
- Take up a lot of programming time.

Disaster: Therac-25

- Medical linear accelerator for cancer treatment (radiation therapy)
- Introduced in 1983; novel use of software control.
- Some patients received massive radiation overdoses, leading to disability and 3 deaths.
- A combination of factors:
  - Software errors
  - Inadequate safety engineering
  - Poor design of user input module
  - Inadequate reporting of incidents

Disasters: NASA Mars Climate Orbiter

- Orbiter's software used Metric units (grams, meters, Celcius, etc)
- Mission control software used Imperial units (lbs, feet, Fahrenheit, etc)
- No one tested whether unit-conversion was being done
- $125M Orbiter crashed into Mars.

Disasters: ESA Cluster (Ariane 5 rocket)

- Safety mechanisms coded in control software for testing
- Safety mechanisms turned off for actual launch
- $370M of high tech destroyed 39 seconds after launch.

Disasters: Scripps Research Retractions

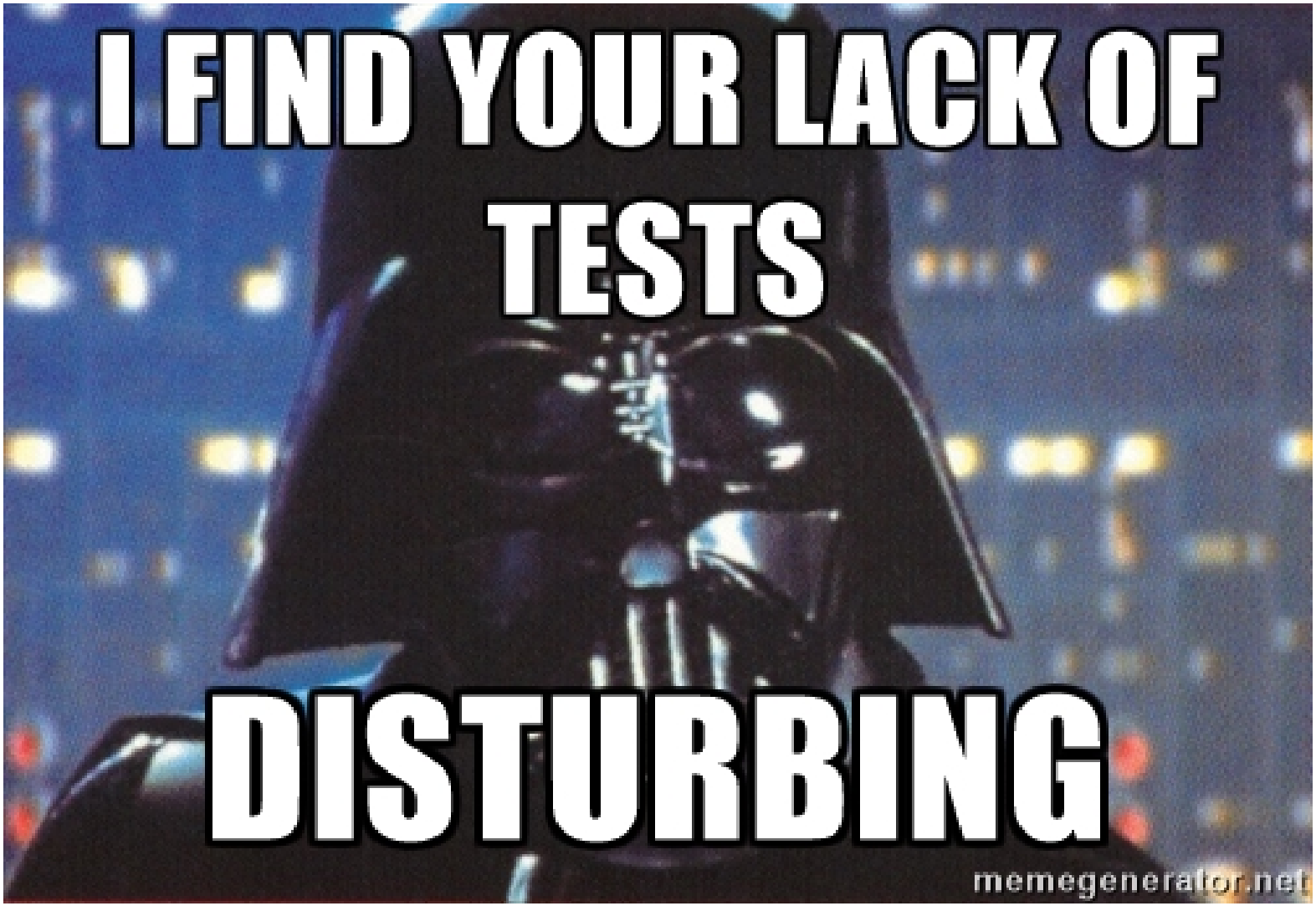
- Data analysis software to deduce protein crystal structure
- Five high ranking publications from the results (Science, JMB, PNAS)
- Researchers discover one function flipped numeric values from positive to negative
- Five high ranking publications retracted (Science, JMB, PNAS)

Why we test software

- Software errors are inevitable.
- Testing is not an after-thought, it is a professional responsibility.
- Testing and debugging has to be part of your development plan.

Software errors will happen

- We're not perfect.
- The sooner we find them and fix them the better.
- Time spent testing will reduce time spent debugging.


I FIND YOUR LACK OF TESTS
DISTURBING
memegenerator.net

Terminology

- Fault: Your program exhibits behaviour that is not intended.
- Error: The cause of the faut.
- Specification: A written document stating the correct behaviour.
- Oracle: A method that tells us if an answer is correct or not.

Testing is the active process of fault discovery.

Where do software faults come from?

- There are two ultimate sources of software errors:
  1. Not writing the code you intended to write.
     - Typographical errors
     - Minor errors in logic.
  2. Not knowing what code to write.
     - Design was bad.
     - No design.
     - No plan.
- Debugging can only help with #1.

Scientific attitude for testing

- Hypothesis: Your program is correct.
- Experiment: Designed to show Hypothesis is false.
  - Try to prove that your own program has errors.
- Conclusion: Confidence in Hypothesis increases with every Experiment.

Questions about testing

- How should we test?
- What should we test?
- When should we test?
- How much time should we spend testing?

These questions are addressed more thoroughly in higher level CMPT courses.

Testing Review: Strategies

- Black-box testing:
  - Design tests based on the interface of a function (inputs and outputs).
  - (black-box means: don't look at the code inside the function)
- White-box testing:
  - Design tests based on the actual instructions inside the function.
  - (white-box means: look at the code inside the function)
- These are not entirely distinct from each other.

Test coverage

- Measures how much of the code has been tested.
- Measured in terms of:
  - Lines of code
  - Number of branches
  - Number of functions
  - Number of modules
- More test coverage is better.

Testing levels

- Unit testing
  - Testing done on code units: e.g., function, method.
- Integration testing
  - Testing done on functions or modules working together.
- System testing
  - Testing done on completed applications.

Degrees of Testing

Testing can be considered at various levels of diligence:

1. No testing
2. Trivial, simple examples only
3. Reasonable, expected examples
4. Difficult examples
5. Unreasonable examples, illegal inputs
6. All possible inputs

Questions about testing

Answered from student perspective

- How should we test?
  white-box, black-box
- What should we test?
  unit, integration, system
- When should we test?
  immediately, or sooner
- How much time should we spend testing?
  equal to time spent coding
  It's not wasted time.

Test case design

- You can't test exhaustively!
- Each test case should be an example drawn from an equivalence class.
- To prevent bias, draw two test cases for every equivalence class.
- Draw test cases from the boundaries of equivalence classes.

Test Case Equivalence Classes

A set of test cases is called an equivalence class if:

- They all cause same path through the code (white-box).
- Or, the result on any test case in the set is the same on all the others (black-box).

An equivalence class expresses a range of possible test cases:

- Numbers: positive, zero, negative, even, odd.
- Lists: empty; singleton; all the same; increasing order; decreasing order; even size; odd size.

Boundary test cases

- An equivalence class expresses a range of possible test cases
- Values on the boundary of the classes are called boundary values
- Include test cases at every boundary, and on both sides.

Test scripts

- Unit testing: An if-statement checking a function's output against a known value.
- Silent unless an error is detected.
- Easy to do when output is simple, e.g., numbers, strings, lists.
- Harder to do with more sophisticated programs, e.g., node-chains, trees

Test Harnesses

- Test Harness: A script or function that initializes data to be used in a test.
  - May have to simulate the effects of other functions to be tested
  - Possibly uses other functions that are being tested together.
- Silent unless an error is detected.
- Test harnesses may have to be tested too!