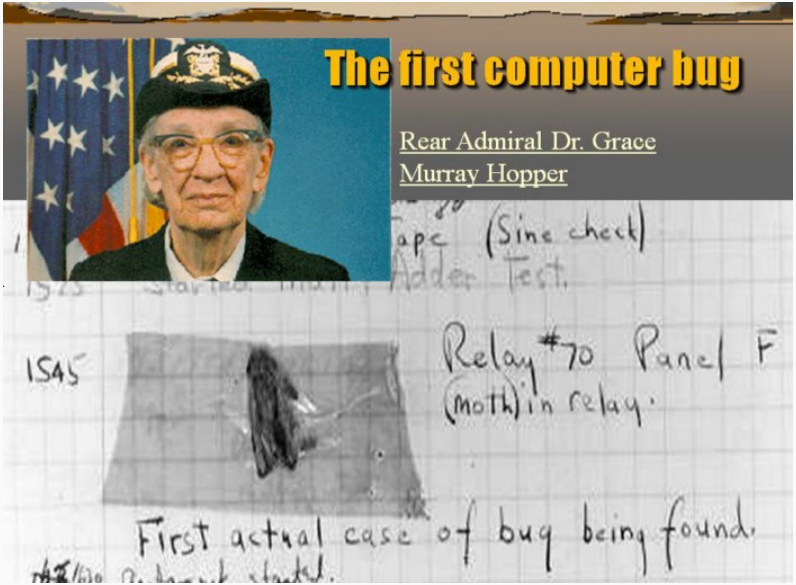
Lecture 5b: Testing

CS 70: Data Structures and Program Development

Thursday, Feb 20, 2019

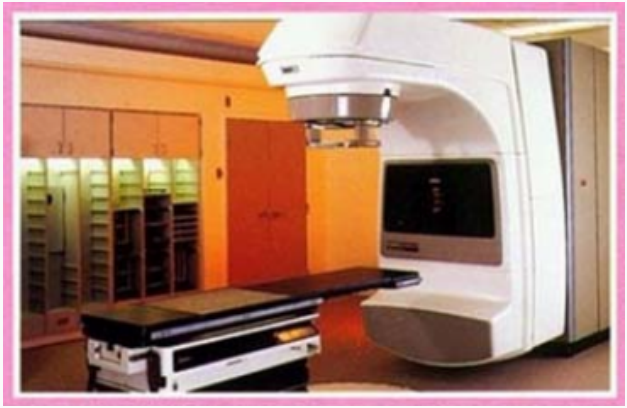
What is this?

What is this?


The first computer bug
Rear Admiral Dr. Grace Murray Hopper
(Sine check)
1545
Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.

Testing

Why test?



Therac 25 radiation therapy machine.

Delivered 100 times safe level of radiation due to software error.

Why test?

WELLS FARGO ADMITS HUNDREDS OF CUSTOMERS LOST HOMES DUE TO SOFTWARE GLITCH

BY DAN CANCIAN ON 8/5/18 AT 9:28 AM

Why test?

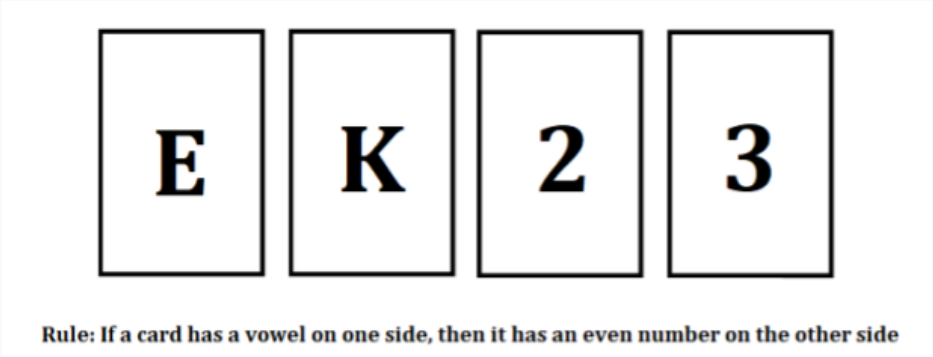
Ariane 5 Rocket Video

Why test?

Ariane 5 Rocket Video

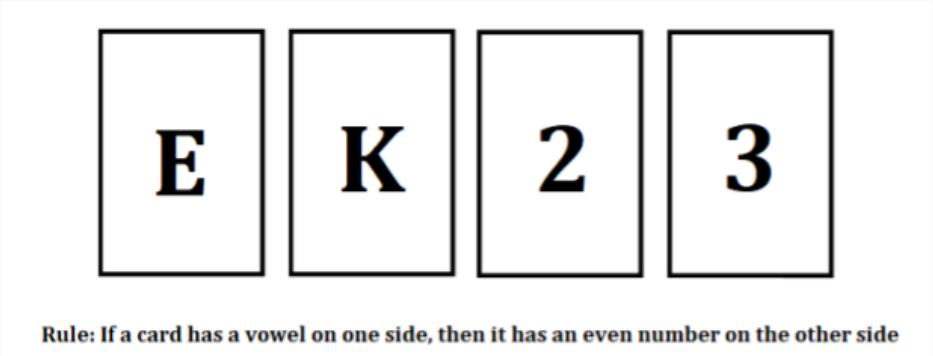
The Testing State of Mind

Let's Play A Card Game



Which cards should we flip over to decide if the rule is true?

Let's Play A Card Game



Which cards should we flip over to decide if the rule is true?

This is called the Wason Selection Task.

90% of those tested did not solve it.

Testing: Philosophy of Science Point of View

"My proposal is based on an asymmetry between verifiability and falsifiability; an asymmetry which results from the logical form of universal statements. For these are never derivable from singular statements, but can be contradicted by singular statements."

- Karl Popper: The Logic of Scientific Discovery 1959

Testing: Philosophy of Science Point of View

"My proposal is based on an asymmetry between verifiability and falsifiability; an asymmetry which results from the logical form of universal statements. For these are never derivable from singular statements, but can be contradicted by singular statements."

- Karl Popper: The Logic of Scientific Discovery 1959

"Program testing can be used to show the presence of bugs, but never to show their absence!""

- Edsger Dijkstra. Notes On Structured Programming 1970

Testing: Philosophy of Science Point of View

"My proposal is based on an asymmetry between verifiability and falsifiability; an asymmetry which results from the logical form of universal statements. For these are never derivable from singular statements, but can be contradicted by singular statements."

- Karl Popper: The Logic of Scientific Discovery 1959

"Program testing can be used to show the presence of bugs, but never to show their absence!""

- Edsger Dijkstra. Notes On Structured Programming 1970

"I don't know how many of you have ever met Dijkstra, but you probably know that arrogance in computer science is measured in nano-Dijkstras."

- Alan Kay

What does that have to do with testing?

What is the purpose of testing?

- To show that the code has no bugs? A nice ideal.
- Is exhaustive testing possible?
- Consider an application with input fields:
  - First Name: up to 20 characters
  - Last Name: up to 20 characters
  - Phone Number: 10 digits
  - Too many inputs to test!

The goal of testing is to *find errors*.

The problem of un-interrogated optimism

Titus Bartik, et. al.: Designing for Dystopia: Software Engineering Research for the Post-apocalypse. (FSE 2016)

- Literary theorists have long recognized the trade-offs in optimistic and pessimistic thinking through utopias and dystopias.
- Research suggests that scientists are overwhelmingly optimistic, and subject to the effect of optimism bias [1].
- Software engineering researchers have a tendency to be optimistic. Though useful, optimism bias bolsters unrealistic expectations towards desirable outcomes.
- Framing software engineering research through dystopias mitigates optimism bias and engender more diverse and thought-provoking research directions.

[1] D. A. Armor and S. E. Taylor. When predictions fail: The dilemma of unrealistic optimism.

Testing Guidelines

Why test during development?

Depends on the project.

- Not necessary in ALL cases
  - small, simple pieces of code
  - code that won't ever be re-used
- Designing an interface for many other users? definitely test.
- In projects where one makes big design choices or there are multiple strategies, testing is especially important.

Aspirationally: tests come first.

Test first?

Test first gives you a hint as to whether your interface suffices.

- Is it enough to solve a problem?

Test first suggests that your interface is even testable.

- Does your code actually have the *potential* to work?

Test first increases your velocity in general.

- You get to your solution quicker
- Get to see how far you have gotten and how much more you have until you reach your goal

What is a test?

A test should have several components:

- The test input or scenario
- The expected result
- Documentation
  - What part of the requirements is being tested?
  - What is the reasoning behind the test?

How do we test?

- Find a framework to make testing easy
  - i.e. the lightweight testing-logger we provided
  - affirm test conditions
- Testing Domain Specific Languages (DSLs)
- C++: gtest (an open source project, stands for Google Test).
- Python: pytest
- Java: junit, Truth

Equivalence Partitioning

What are some good test values for this function?

```
double compute(double x){
  double y = x + 1.0;
  if(y < 10.0){
    return y;
  }
  return 10;
```

Equivalence Partitioning

What are some good test values for this function?

```
double compute(double x){
  double y = x + 1.0;
  if(y < 10.0){
    return y;
  }
  return 10;
```

Two equivalence partitions for x

- x $<$ 9
- x $>=$ 9

We can choose just a single representative from each partition

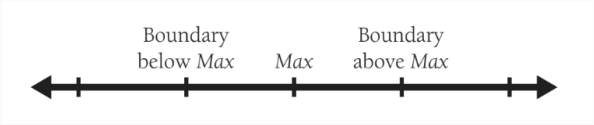
- x is 5
- x is 15

Boundary Analysis

Extremely common mistakes:

- you write *num* when you meant *num - 1*
- you write $>=$ when you meant $>$

Explicitly write tests to discover these types of errors.

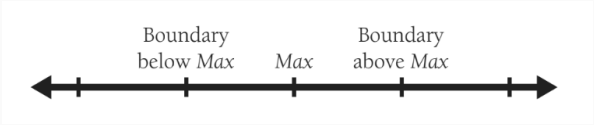


Boundary Analysis

Extremely common mistakes:

- you write *num* when you meant *num - 1*
- you write $>=$ when you meant $>$

Explicitly write tests to discover these types of errors.




```
double compute(double x){
  double y = x + 1.0;
  if(y < 10.0){
    return y;
  }
  return 10;
```

We should test the boundaries:

- x is 9, x is 10, x is 11

Explicitly test for bad data

Examples:

- Too little data:
  - an empty vector
- Too much data:
  - array of 1 million employees
- Invalid data:
  - Negative student ID number

What about uninitialized data??

```
int x;
compute(x);
```

Explicitly test for good data

Examples:

- Nominal data: middle of the road, expected data
  - e.g. 10 employees in database
- Minimum nominal configuration
  - e.g. 1 employee in database
- Maximum nominal configuration
  - e.g. 1,000 employees in database

Unit testing

Test individual units of code:

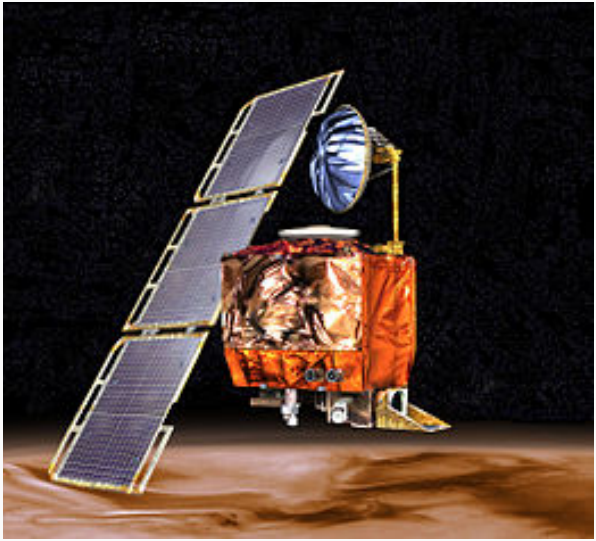
- typically, test every single function

How much testing is enough?

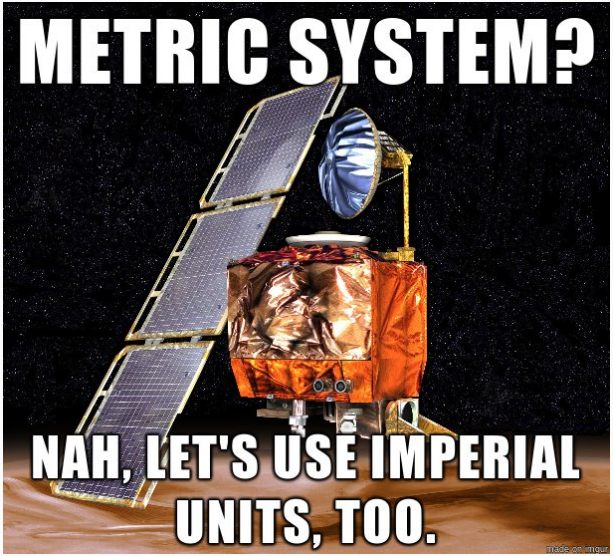
At a *minimum*

- Cover every statement
- Cover every branch

Mars Climate Orbiter ($327.6 million)



Mars Climate Orbiter ($327.6 million)



Integration testing

After individual units (e.g. functions) are tested, do they correctly work together?

Two main ways to combine units:

Output of one passed to the other

```
int x = function1(y);
int z = function2(x);
```

One function is called by another

```
int function3(int x){
  int y = function4(x)
  return y;
}
```

Integration testing with a Driver

For integration testing, a driver is a function or class whose sole purpose is to combine two or more units.

```
int driver(int y){
  x = function1(y);
  z = function2(x);
  return z;
}
```

A Little Practice

Consider an application with input fields:

* First Name: up to 20 characters
* Last Name: up to 20 characters
* Phone Number: 10 digits
* Too many inputs to test!

Come up with some test inputs

3 Goals of Good Tests

- **Fidelity**: Some test should fail if you make a breaking change in your code.

Maximize fidelity by ensuring that your tests cover all the paths through your code and include all relevant assertions.

- **Resilience**: Your test only fails if you make a breaking change.

Maximize resilience by only testing the interface, not the internal details.

- **Precision**: If your test fails, it tells you (exactly) what failed.

Maximize precision by keeping tests small and tightly focused (avoid relying on large end-to-end tests).

E.g., for a chess game, test small pieces like "is the king in check for this setup" rather than "does white win after a game is played".

Summary

- Testing is for finding bugs! **Try** to break your own code.
- Unit tests: test individual functions.
- Integration tests: test composition of functions.
- Don't test undefined behavior.
- Write tests that have high fidelity, resilience, and precision.