Program Validation and Testing

José Costa

Software for Embedded Systems

Departamento de Engenharia Informática (DEI)
Instituto Superior Técnico

2015-10-26

Outline

- Program Validation and Testing
- Software Can Fail
- Basic Testing
- Black-box / Clear-box Testing
- Controllability and Observability
- Path Testing
- Branch Testing
- Other Methods

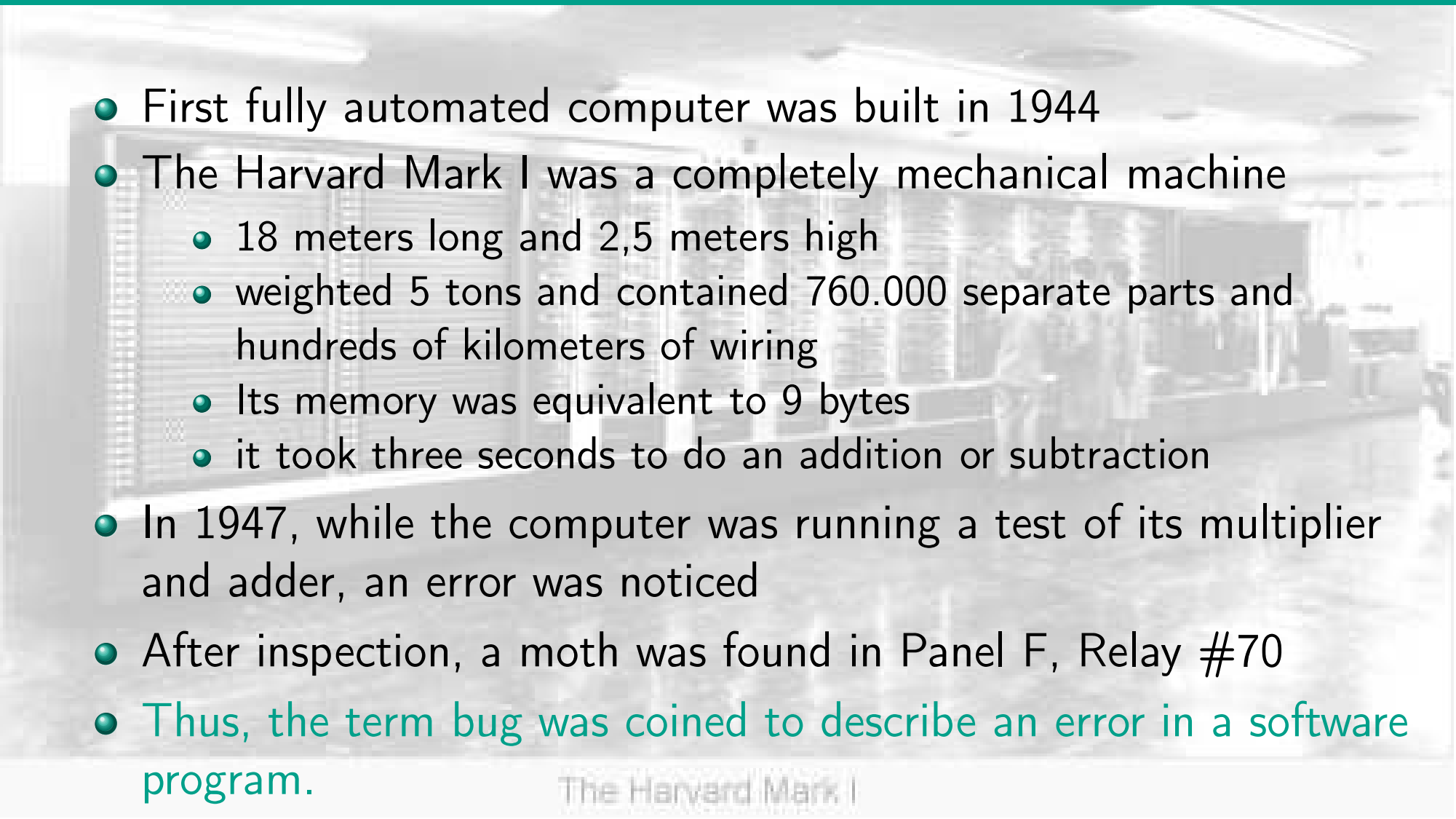
Validation and Testing

TÉCNICO LISBOA

Why is it important to validate and test the software?

Early history

- First fully automated computer was built in 1944
- The Harvard Mark I was a completely mechanical machine
  - 18 meters long and 2,5 meters high
  - weighted 5 tons and contained 760.000 separate parts and hundreds of kilometers of wiring
  - Its memory was equivalent to 9 bytes
  - it took three seconds to do an addition or subtraction
- In 1947, while the computer was running a test of its multiplier and adder, an error was noticed
- After inspection, a moth was found in Panel F, Relay #70
- Thus, the term bug was coined to describe an error in a software program.


The Harvard Mark I

- Since then we have evolved...

Software Can Fail

Software Can Fail

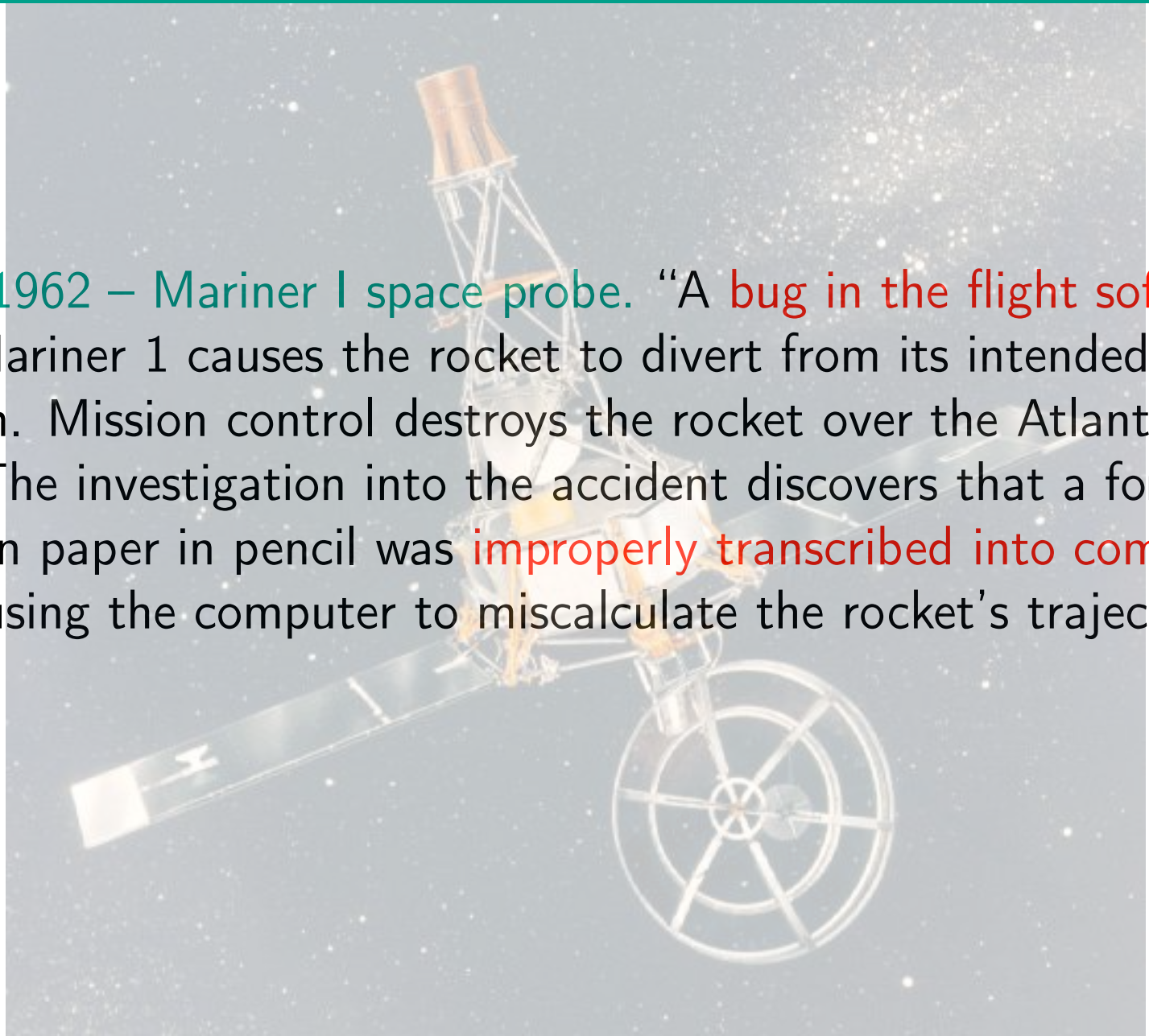
Software Can Fail

Software Can Fail

Software Can Fail

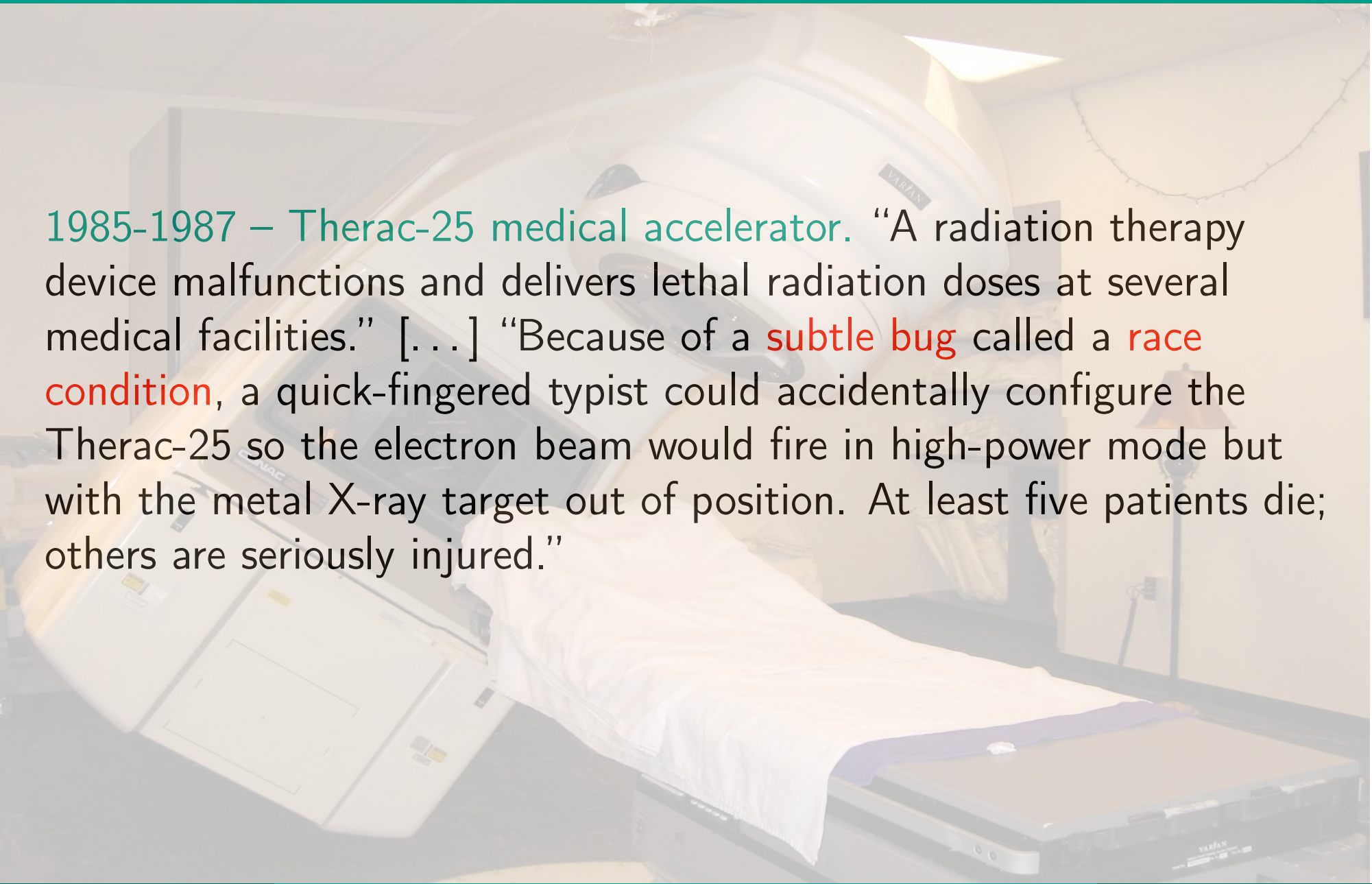
Software Can Fail

Some Serious Failures



July 28, 1962 – Mariner I space probe. "A bug in the flight software for the Mariner 1 causes the rocket to divert from its intended path on launch. Mission control destroys the rocket over the Atlantic Ocean. The investigation into the accident discovers that a formula written on paper in pencil was improperly transcribed into computer code, causing the computer to miscalculate the rocket's trajectory."

Some Serious Failures



1985-1987 – Therac-25 medical accelerator. "A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities." [...] "Because of a subtle bug called a race condition, a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others are seriously injured."

Some Serious Failures

June 4, 1996 – Ariane 5 Flight 501. "Working code for the Ariane 4 rocket is reused in the Ariane 5, but the Ariane 5's faster engines trigger a bug in an arithmetic routine inside the rocket's flight computer. The error is in the code that converts a 64-bit floating-point number to a 16-bit signed integer. The faster engines cause the 64-bit numbers to be larger in the Ariane 5 than in the Ariane 4, triggering an overflow condition that results in the flight computer crashing. First Flight 501's backup computer crashes, followed 0.05 seconds later by a crash of the primary computer. As a result of these crashed computers, the rocket's primary processor overpowers the rocket's engines and causes the rocket to disintegrate 40 seconds after launch."

Some Serious Failures

November 2000 – National Cancer Institute, Panama City.

"Multidata's software allows a radiation therapist to draw on a computer screen the placement of metal shields called "blocks" designed to protect healthy tissue from the radiation. But the software will only allow technicians to use four shielding blocks, and the Panamanian doctors wish to use five. The doctors discover that they can trick the software by drawing all five blocks as a single large block with a hole in the middle. What the doctors don't realize is that the Multidata software gives different answers in this configuration depending on how the hole is drawn: draw it in one direction and the correct dose is calculated, draw in another direction and the software recommends twice the necessary exposure. At least eight patients die, while another 20 receive overdoses likely to cause significant health problems. The physicians, who were legally required to double-check the computer's calculations by hand, are indicted for murder."

Some Serious Failures

TÉCNICO LISBOA

2010 - Toyota recall. "The recalls was related to issues with the Prius's brakes, which Toyota said was caused by a software glitch. The company said it was looking into the best way to solve the problem. An internal NHTSA memo indicated that the issue was the short delay in regenerative braking when hitting a bump, resulting in increased stopping distance. Toyota officials described the problem as a "disconnect" in the vehicle's complex anti-lock brake system (ABS) that causes less than a one-second lag. With the delay, a vehicle going 100km/h will have traveled nearly another 30 m before the brakes begin to take hold."

Cost of Bugs

- Software bugs cost $59,5 thousand million a year
- An estimated $22,2 thousand million could be eliminated
  - by improved testing that enables earlier and more effective identification and removal of defects
- More than half of the errors are not found until later in the development process or during post-sale use of software.

2002 Study by U.S. Department of Commerce National Institute of Standards and Technology (NIST)

- Finding and fixing coding problems costs software makers and the global economy $312 thousand million a year
- Software developers spend 50% of their programming time finding and fixing bugs

2013 Cambridge University/Undo Software/Rogue Wave Software research study

Goals of Testing

- Make sure software works as intended
  - we will concentrate here on functional testing
- How do we do that?
- What tests are required to adequately test the program?
- What is “adequate”?

Basic Testing Procedure

1. Provide the program with inputs
2. Execute the program
3. Compare the outputs to expected results

Types of Software Testing

Black-box

Tests are generated without knowledge of program internals

Clear-box (white-box)

Tests are generated from the program structure

Black-box Testing

- Black-box tests are made from the specifications, not the code

- Black-box testing complements clear-box
  - probably may test unusual cases better

Types of Black-box Tests

- Specified inputs/outputs
  - select inputs from spec, determine required outputs
- Random
  - generate random tests, determine appropriate output
- Regression
  - tests used in previous versions of system

Clear-box Testing

- Generate tests based on the structure of the program
- Is a given block of code executed when we think it should be executed?
- Does a variable receive the value we think it should get?

Controllability and Observability

Controllability

Must be able to cause a particular internal condition to occur.

Observability

Must be able to see the effects of a state from the outside.

Example: FIR Filter

Example (FIR filter)

```
for (firout = 0.0, j = 0; j < N; j++)
  firout += buff[j] * c[j];
if (firout > 100.0) firout = 100.0;
if (firout < -100.0) firout = -100.0;
```

Controllability

- to test range checks for `firout`, must first load circular buffer

Observability

- how do we observe values of `buff`, `firout`?

Path-based Testing

- Clear-box testing generally tests selected program paths
  - control program to exercise a path
  - observe program to determine if path was properly executed

- May look at whether location on path was reached (control), whether variable on path was set (data)

Points of View

There are several ways to look at control coverage

Coverage metrics

- Statement coverage
- Path coverage
  - basis sets
  - cyclomatic complexity
- Branch testing
- Domain testing

How Do We Choose the Paths

Two possible criteria for selecting a set of paths

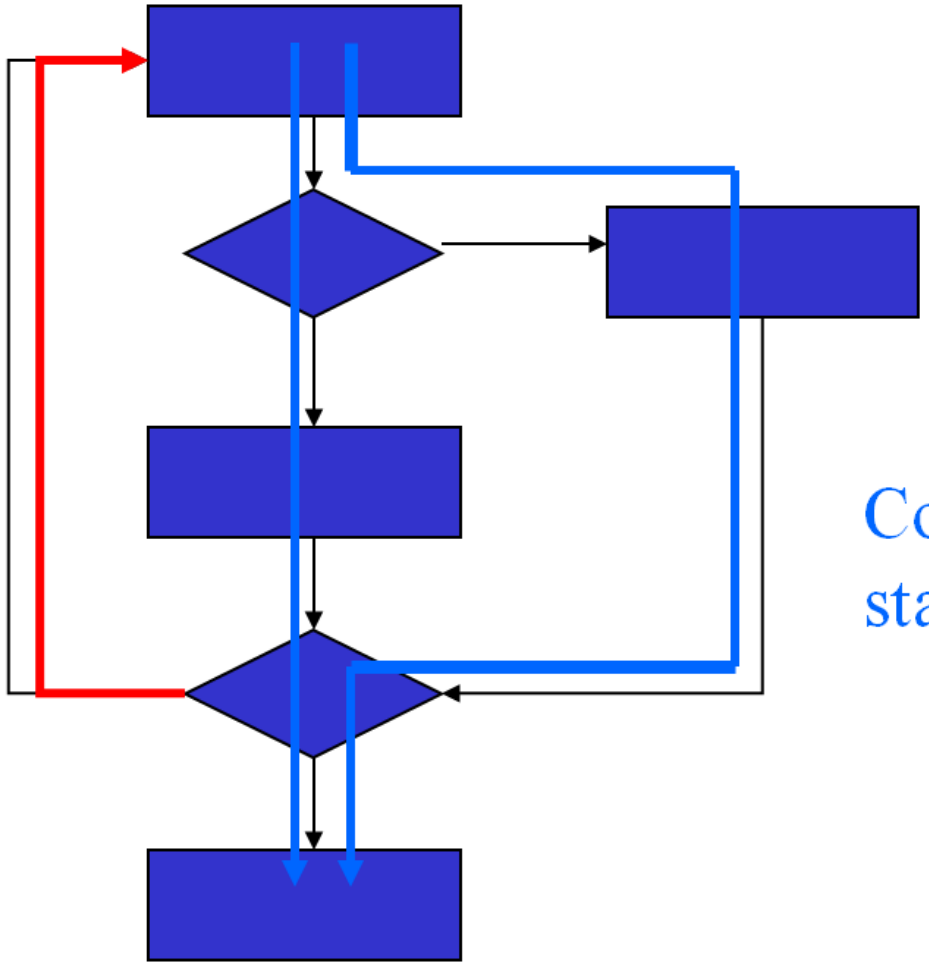
- Execute every statement at least once
- Execute every direction of a branch at least once

Other criteria

- Try to reach a particular statement
  - e.g., an assert

Path Example

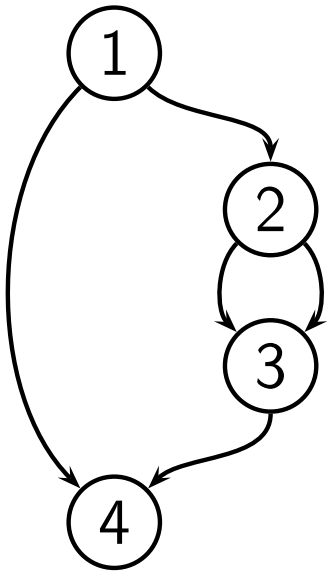
+/+ Covers all branches



Covers all statements

Cyclomatic Complexity

- Provides an upper bound on the control complexity of a program
- Cyclomatic complexity: $M = e - n + 2p$
  - e = # edges in control graph
  - n = # nodes in control graph
  - p = # graph components



- Complexity of a structured program: # binary decisions + 1
  - Add `b-1` for each b-way branch

Branch Testing Strategy

Branch testing

- Exercise the elements of a conditional
  - not just one true and one false case
  - based on the structure of the condition

Strategy

Devise a test for every simple condition in a Boolean expression

Example: Branch Testing

Example (Branch testing)

- Meant to write:

```
if (a || (b >= c)) { printf("OK\n"); }
```

- Actually wrote:

```
if (a && (b >= c)) { printf("OK\n"); }
```

Branch testing strategy

- One test is a =False, $(b >= c)$ = True
- If $a = 0, b = 3, c = 2$
  - produces different answers

Another Branch Testing Example

Example (Branch testing)

- Meant to write:

```
if ((x == good_pointer) && (x->field1 == 3))...
```

- Actually wrote:

```
if ((x = good_pointer) && (x->field1 == 3))...
```

Branch testing strategy

- If we use only `field1` value to exercise branch, we may miss pointer problem

Domain Testing

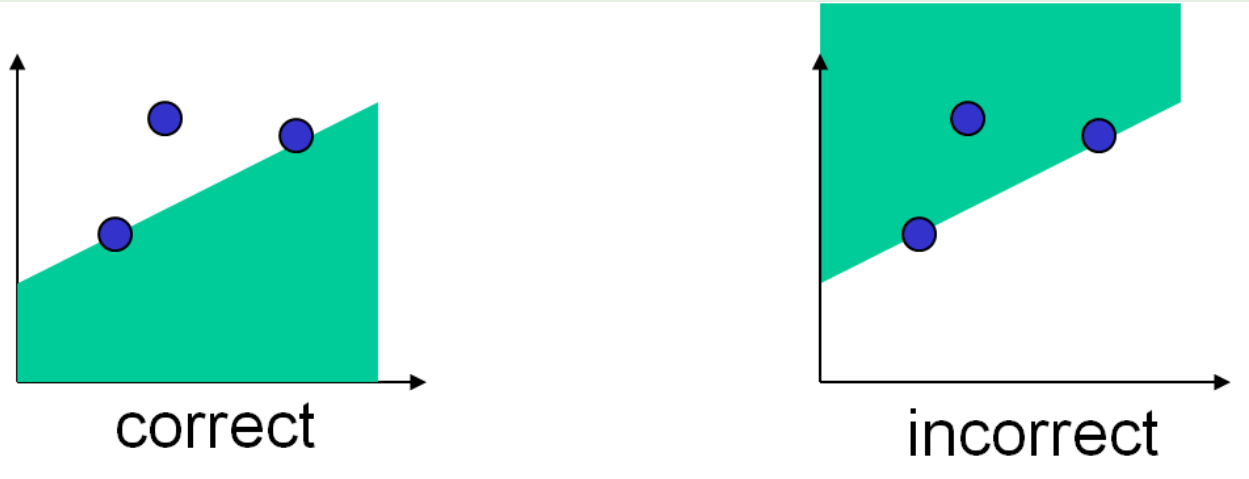
Domain

A set of inputs for which the program executes the same path

- Concentrates on linear inequalities

Example (condition j <= i + 1)

- Test two cases on boundary, one outside boundary



Def-use analysis

Match variable definitions (assignments) and uses.

Example (Def-use)

```
x = 5;
...
if (x > 0) ...
```

- Does assignment get to the use?
- Must devise test that assigns and uses variable `x`

Loop Testing

- Loops are a common, specialized structure - specialized tests can help

Example (Useful test cases)

- skip loop entirely
- one iteration
- two iterations
- mid-range of iterations
- n-1, n, n+1 iterations

- What about nested loops?

Verification and Formal Methods

Formal methods use a mathematical specification of the system

Advantages

- Using the mathematical specification they can uncover all bugs
- Can be used alongside the specification to detect implementation errors (before testing)

Disadvantages

- Very costly in terms of computational power and human labour
- Same level of confidence can be reached with other more cheaply techniques

Evaluating Tests

- How good are your tests?
- How do you know that they are good?

Compare bugs found

Keep track of bugs found, compare to historical trends

Error injection

Add bugs to copy of code, run tests on modified code.

Outline

- Program Validation and Testing
- Software Can Fail
- Basic Testing
- Black-box / Clear-box Testing
- Controllability and Observability
- Path Testing
- Branch Testing
- Other Methods

References

Computers as Components: Principles of Embedded Computing System Design , Marylin Wolf. Morgan Kaufmam. Cap 5.9

Next Class

- Accelerators