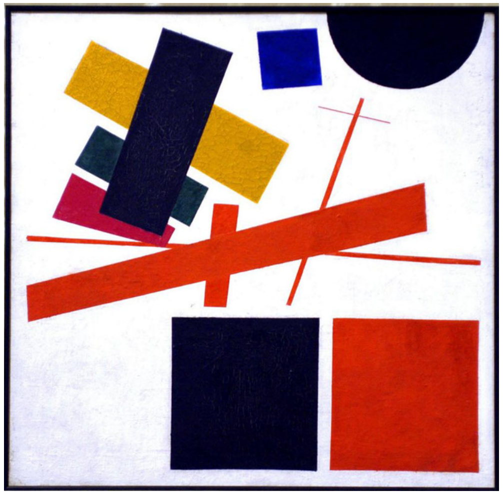
CPPEN 422
Software Testing and Analysis



Lecture 03

Given that:

- Each test case is one executable **example** of system behavior
- Either functional or structural

Identify 3 main testing challenges

Discuss with your neighbor

2.2
10.2

Testing: An easy job?

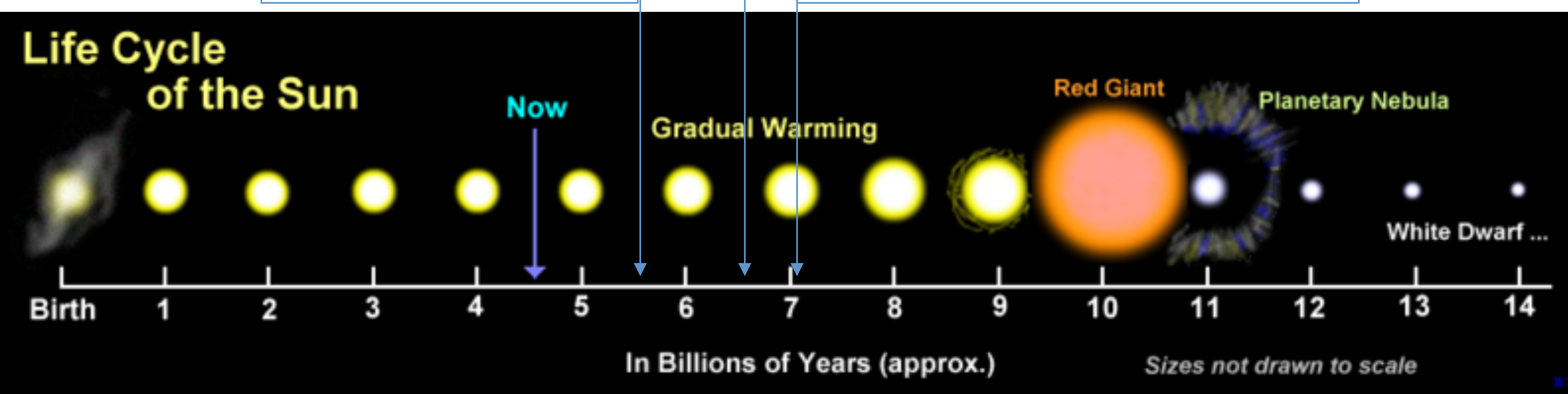
- A simple program:
  3 inputs, 1 output
- a,b,c: 32 bit integers
- trillion test cases / s.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

All oceans dry

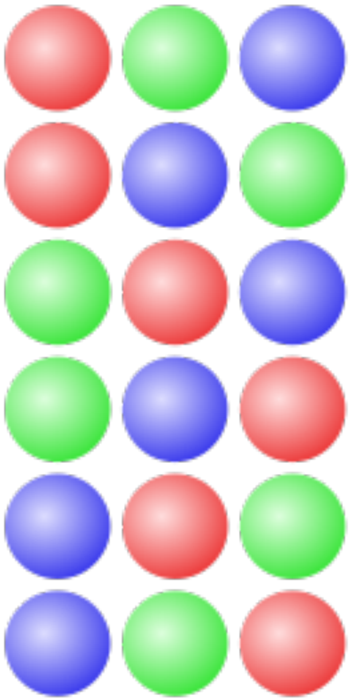
All plants dead

Tests done (2.5 bill. y)



Testing Challenge: Too Many Permutations

- 1 trillion test cases to test a formula with 3 inputs, a, b, c.

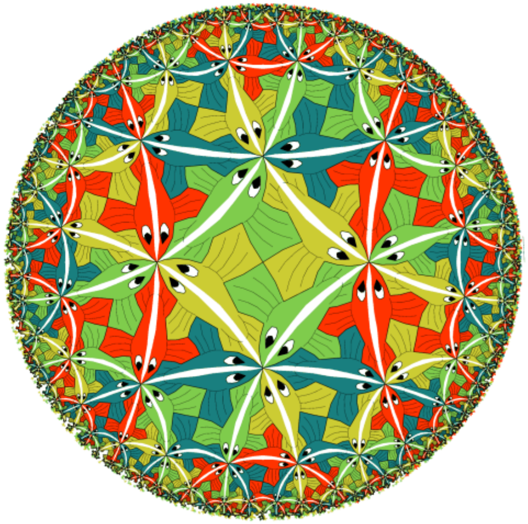


Testing Challenge: Execution Time

- 1 trillion test cases
- 1 test case takes 1 second
- 1.5 billion years to execute the test cases

The Set of Examples is Incomplete

- Too much data
- Too many combinations
- Too many paths



- Covering all properties of interest (i.e. all examples) is fundamentally **undecidable**

Analysis versus Testing

- Does (desirable) property *P* hold for system S?
  - P = "Does S terminate"?

- **Software Testing: *Optimistic Accuracy***
  - S terminates on 4 inputs: it will always terminate.

- **Static analysis: *Pessimistic Accuracy***
  - We can only be sure S terminates if it doesn't contain a "while" or other looping construct
  - S contains a while, hence it doesn't terminate.

Testing Challenge: Observability & Controllability

17.3

Observability:

- Can I see what's going on "inside"?

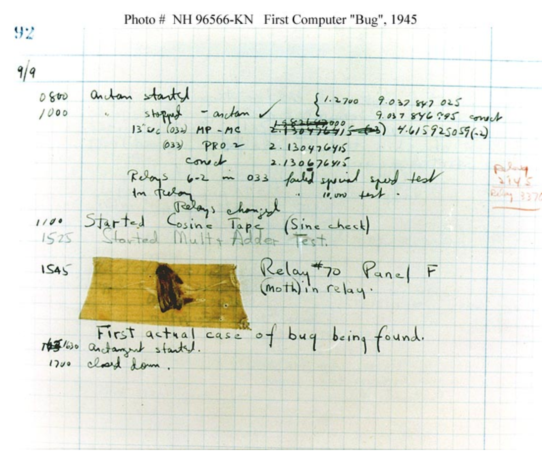
Controllability:

- Can I influence what's happening?

Testing Challenge: Faults May Hide

- **Failure:**
  - Manifested inability of a system to perform required function.
- **Fault:**
  - Incorrect/missing code
- **Error**:
  - human action producing fault
- **And thus**:
  - Testing is: triggering failures
  - Debugging: finding faults given a failure

"bug"



Challenge: RIP
Reach, Infect/Trigger, Propagate

3.1

You *reach* the fault only if y > 3

x * x should be x + x.
x = 0 or x = 2 won't trigger the fault

Fault only propagated if x * x <= 4 and y < 6 (x=1 and y=4 or 5).

Faults may hide!

x * x should be x + x

```
function foo(x, y) {
 if (y > 3) {
   z = x * x; \\fault
   if (z<=4 && y<6) {
     return( z );
   }
 }
}
```

3.1

Challenge: RIP
Reach, Infect/Trigger, Propagate

Typo:
x * x should be: x + x

```
function foo(x, y) {
 if (y > 3) {
   z = x * x; \\fault
   if (z<=4 && y<6) {
      return( z );
   }
 }
}
```

```
@Test
void testFoo () {
 int r = foo(nr1, nr2);
 assertThat(r, exp_nr);
}
```

```
foo(x, y) {
 if (y > 3) {
   z = x * x; \\fault
   if (z<=4 && y<6) {
      return( z );
   }
 }
}
```

```
@Test
void testFoo1() {
 int r = foo(2, 1);
 assertThat(r, 4);
}
```

```
foo(x, y) {
 if (y > 3) {
   z = x * x; \\fault
   if (z<=4 && y<6) {
      return( z );
   }
 }
}
```

```
@Test
void testFoo2() {
 int r = foo(2, 3);
 assertThat(r, 4);
}
```

```
foo(x, y) {
 if (y > 3) {
   z = x * x; \\fault
   if (z<=4 && y<6) {
      return( z );
   }
 }
}
```

```
@Test
void testFoo3() {
 int r = foo(2, 5);
 assertThat(r, 4);
}
```

2 * 2 = 4 but so is
2 + 2 = 4 so the fault is **NOT** detected

```
foo(x, y) {
 if (y > 3) {
   z = x * x; \\fault
   if (z<=4 && y<6) {
      return( z );
   }
 }
}
```

```
@Test
void testFoo3() {
 int r = foo(1, 5);
 assertThat(r, 2);
}
```

1 * 1 = 1 but
1 + 1 = 2 so the fault is detected

```
foo(x, y) {
 if (y > 3) {
   z = x * x; \\fault
   if (z<=4 && y<6) {
      return( z );
   }
 }
}
```

In-class Exercise

```
// Effects: If x==null throw Null Exception
// else return the number of positive elements in x

public int countPositive(int[] x) {
  int count = 0;
  for (int i=0; i<x.length; i++) {
    if (x[i] >= 0) {
      count++;
    }
  }
  return count;
}
```

(a) Identify the **fault**.

(b) Write a test case that does not **execute** the fault.

(c) Write a test case that executes the fault, but does not result in a **detectable error state**.

(d) Write a test case that **detects the fault**.

(a) The fault

The fault is **=> 0 inside the if**. The correct check must have been:

if (x[i] > 0) {

```
public int countPositive(int[] x) {
  int count = 0;
  for (int i=0; i<x.length; i++) {
    if (x[i] >= 0) {
      count++;
    }
  }
  return count;
}
```

(b) Write a test case that does not **execute** the fault.

x must be either null or empty. All other inputs result in the fault being executed. We give the empty case here.

Input: x = []

Expected Output: 0, Actual Output: 0

```
public int countPositive(int[] x) {
  int count = 0;
  for (int i=0; i<x.length; i++) {
    if (x[i] >= 0) {
      count++;
    }
```

(c) not result in a **detectable error state**.

Any **nonempty** x without a 0 entry works fine.

Input: x = [1, 2, 3]

Expected Output: 3, Actual Output: 3

```
public int countPositive(int[] x) {
  int count = 0;
  for (int i=0; i<x.length; i++) {
    if (x[i] >= 0) {
      count++;
    }
```

(d) **detects the fault**.

(d) Input: x = [-4, 2, 0, 2]

Expected Output: 2, Actual Output: 3

First Error State: i = 2; count = 1;

```
public int countPositive(int[] x) {
  int count = 0;
  for (int i=0; i<x.length; i++) {
    if (x[i] >= 0) {
      count++;
    }
  }
  return count;
}
```

Solutions

(a) The fault is **=> 0 inside the if**. The correct check must have been:

if (x[i] > 0) {

(b) x must be either null or empty. All other inputs result in the fault being executed. We give the empty case here.

Input: x = []
Expected Output: 0, Actual Output: 0

(c) Any **nonempty** x without a 0 entry works fine.

Input: x = [1, 2, 3]
Expected Output: 3, Actual Output: 3

(d) Input: x = [-4, 2, 0, 2]

Expected Output: 2, Actual Output: 3

First Error State:

  i = 2;

  count = 1;

Given that:

- Each test case is one executable **example** of system behavior
- Either functional or structural

Identify 3 main testing challenges

Discuss with your neighbor

Testing Challenges

- Explosion of **input** combinations
- Explosion of **path** combinations
- Most interesting properties ***undecidable***
- Faults may be hard to reach (**RIP**)
- Faults may **hide** (observability)
- Faults may be hard to **execute** at will (controllability)
- Collected data & the external environment

Why do we test? What is the goal of software testing in an organization?

Testing Goals Based on Test Process Maturity

- Level 0: There's no difference between testing and debugging
- Level 1: The purpose of testing is to show **correctness**
- Level 2: The purpose of testing is to show that the software **doesn't work**
- Level 3: The purpose of testing is not to prove anything specific, but to **reduce the risk** of using the software
- Level 4: Testing is a mental **discipline** that helps all IT professionals develop higher **quality** software

Level 0 Thinking

- Testing is the **same** as debugging

- Does <u>not</u> distinguish between incorrect behavior and mistakes in the program

- Does <u>not</u> help develop software that is reliable or safe

Level 1 Thinking

- Purpose is to show **correctness**
- Correctness is impossible to achieve
- What do we know if no failures?
  - Good software or bad tests?
- Test engineers have no:
  - Strict goal
  - Real stopping rule
  - Formal test technique
  - Test managers are powerless

Level 2 Thinking

- Purpose is to show **failures**
- Looking for failures is a *negative* activity
- Puts testers and developers into an adversarial relationship
- What if there are no failures?

This describes most software companies.

How can we move to a *team approach* ??

Level 3 Thinking

"Testing can only show the **presence of failures, not their absence.**"

- Whenever we use software, we incur some risk
- Risk may be small and consequences unimportant
- Risk may be great and consequences catastrophic
- Testers and developers cooperate to reduce risk

This describes a few "enlightened" software companies

Level 4 Thinking

A mental discipline that increases quality

- Testing is only **one way** to increase quality
- Test engineers can become technical leaders of the project
- Primary responsibility to measure and improve software quality, from the inception of the project

Where Are You?

Are you at level 0, 1, or 2 ?

Is your organization at work at level 0, 1, or 2 ?

or 3 or 4?

We hope to teach you to become "change agents" in your workplace …

Advocates for level 4 thinking