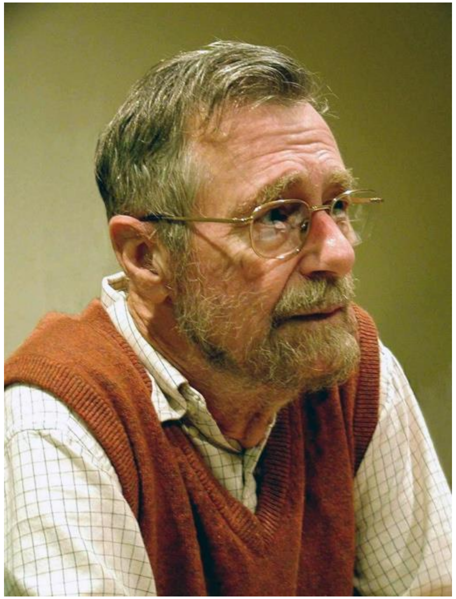
CMSC 433
Programming Language Technologies and Paradigms

Testing

Edsger W. Dijkstra



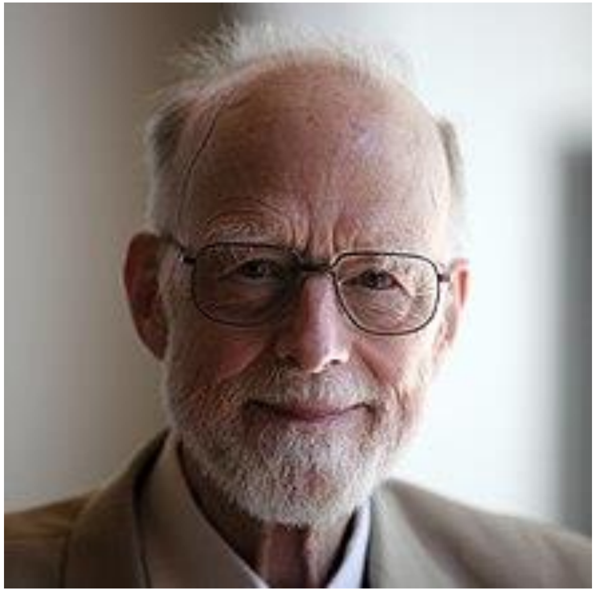
Program testing can be used to show the presence of bugs, but never to show their absence!

“Software testers always go to heaven; they’ve already had their fair share of hell.”

(*Anonymous*)

Tony Hoare

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.



Simple Hashmap

```
let empty v = fun _-> 0;;
let update m k v = fun s->if k=s then v else m s

let m = empty 0;;
let m = update m "foo" 100;;
let m = update m "bar" 200;;
let m = update m "baz" 300;;
m "foo";; (* 100 *)
m "bar";; (* 200 *)
let m = update m "foo" 101;;
m "foo";; (* 101 *)
```

Testing is important

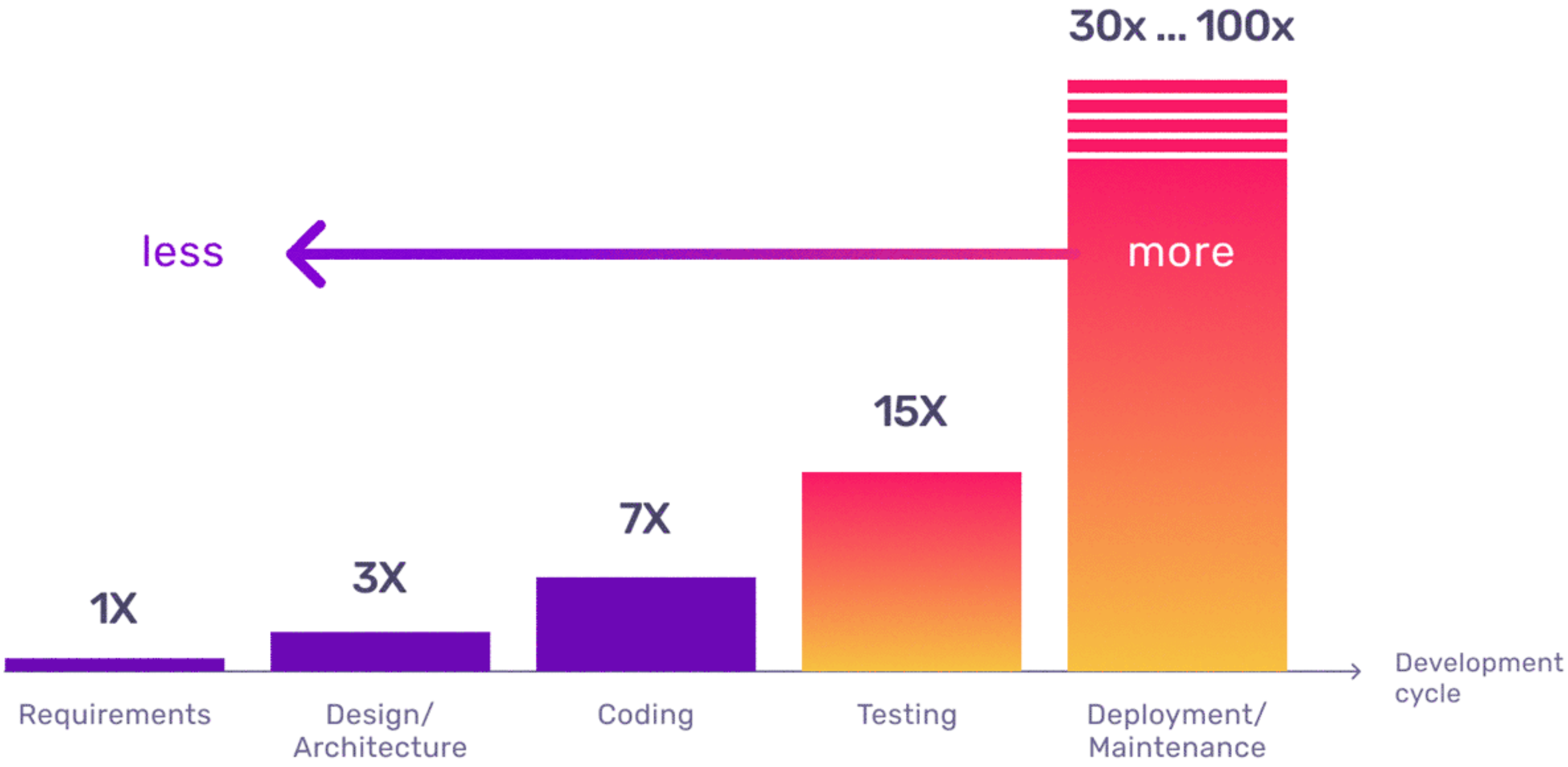
- Estimated **50%** of programmers time spent on finding and fixing bugs.

- Testing is not the only, but the primary method that industry uses to evaluate software under development.

Testing is important

- Ideas and techniques of testing have become essential knowledge for all software developers.
- Expect to use the concepts presented here many times in your career.
- A few basic software testing concepts can be used to design tests for a large variety of software applications.

Cost of Defects

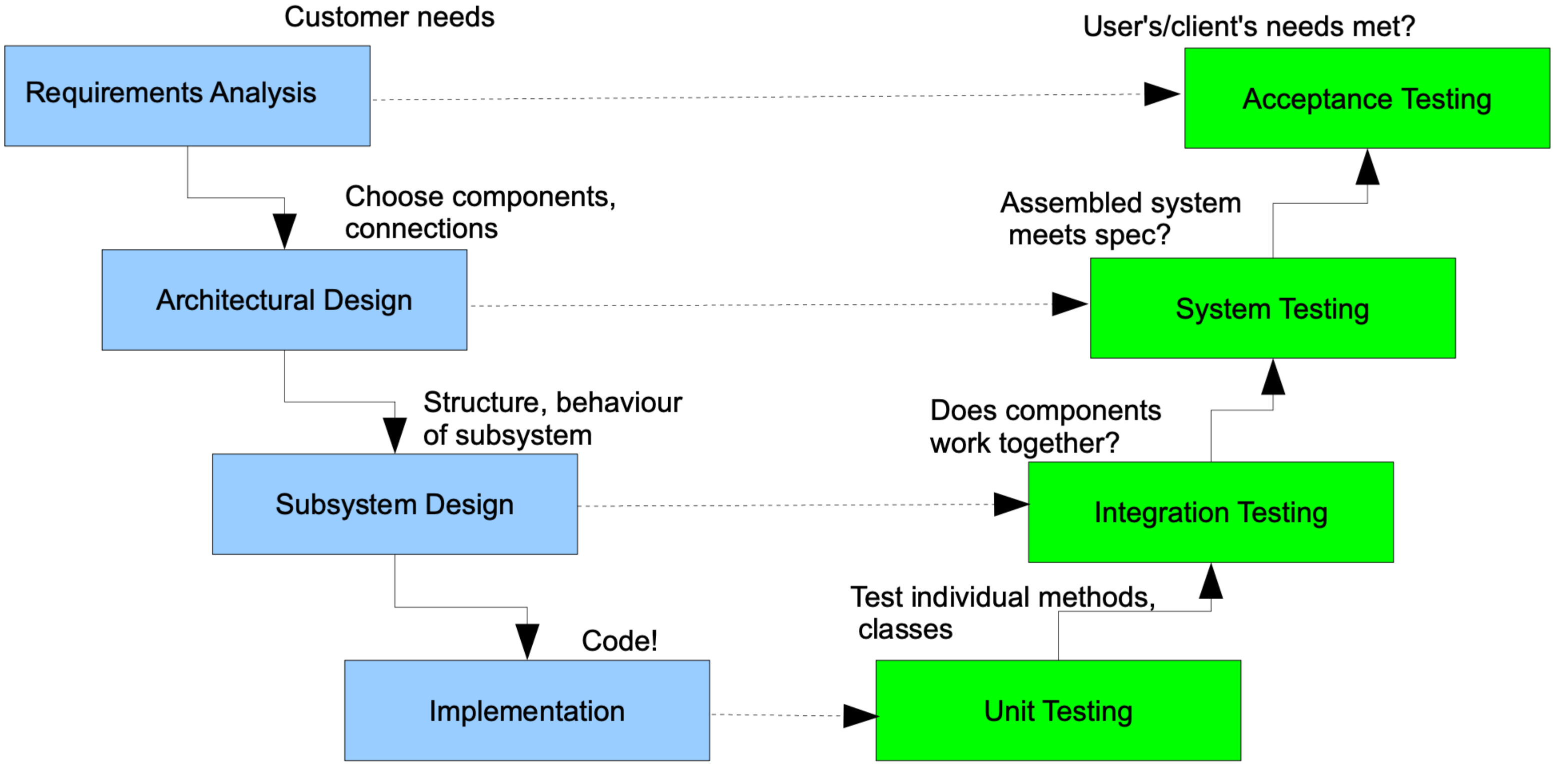


Testing Scale

- Unit testing: testing individual classes/functions
- Integration Testing: testing packages/ subsystems
- System tests: testing the entire system

Unit Test Example: https://github.com/cedar-policy/cedar/blob/main/cedar-policy-core/src/evaluator.rs

V Model



Customer needs

Requirements Analysis

Choose components, connections

Architectural Design

Structure, behaviour of subsystem

Subsystem Design

Code!

Implementation

User's/client's needs met?

Acceptance Testing

Assembled system meets spec?

System Testing

Does components work together?

Integration Testing

Test individual methods, classes

Unit Testing

There are many variants

Testing Process

- Test first: Test driven development (TDD)
  - Write tests before the code
  - Write the code to pass the test
- Test after
  - Check whether existing code passes the tests
- Iteration
  - Retesting
  - Refactoring

Testing: Purpose

- Functional testing
- Performance Testing
- Security testing
- Usability testing
- Availability testing

Property-based Testing

- a framework that repeatedly generates random inputs, and uses them to confirm that properties hold

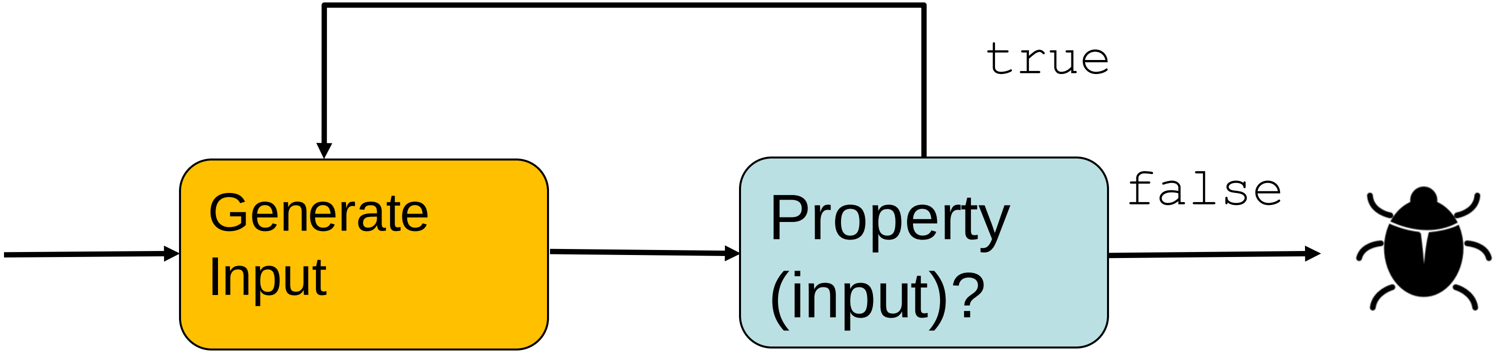
```
public void testList(List<String> l1) {
   List<String> l2 = l1.stream().collect(Collectors.toList());
   Collections.reverse(l2);
   Collections.reverse(l2);
   assertEquals(l1, l2);
}
```

Confirm the property holds for the given input

Repeatedly generate input randomly

QuickCheck: Property-Based Testing

- QCheck tests are described by
  - A generator: generates random input
  - A property: **`bool`**-valued function



Fuzz Testing

- Fuzz testing is a quality assurance technique used to discover coding errors and security loopholes in software, operating systems or networks.

- It involves inputting massive amounts of random data, called fuzz, to the test subject in an attempt to make it crash.

- If a vulnerability is found, a software tool called a fuzzer can be used to identify potential causes.

Mutation Testing

- Mutation testing involves modifying a program in small ways.

```
if (a && b)
   { c = 1; }
else
{ c = 0; }
```

The condition mutation operator would replace && with || and produce the following mutant:

```
if (a || b)
   { c = 1; }
else
   { c = 0; }
```

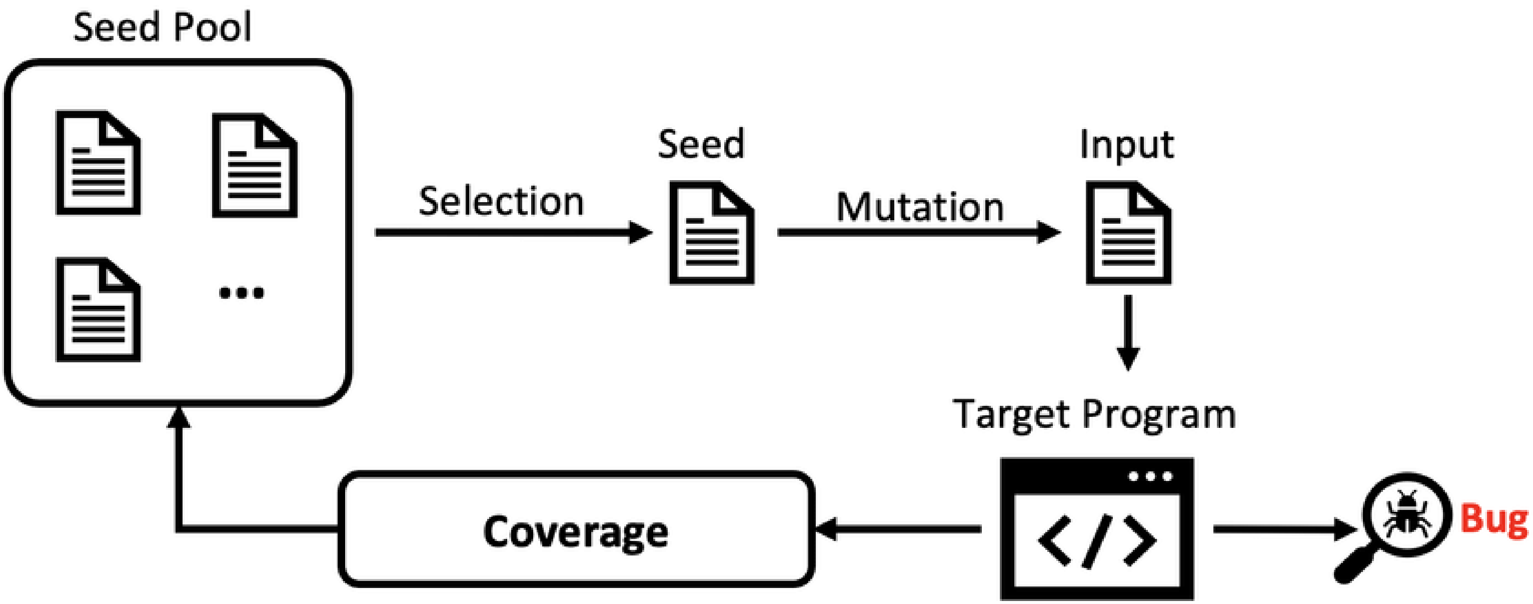
Mutation Operators

- Many mutation operators have been explored by researchers. Here are some examples of mutation operators for imperative languages:
  - Statement deletion
  - Statement duplication or insertion, e.g. goto fail;
  - Replacement of boolean subexpressions with *true* and *false*
  - Replacement of some arithmetic operations with others, e.g. + with *, - with /
  - Replacement of some boolean relations with others, e.g. > with >=, == and <=
  - Remove method body
  - …

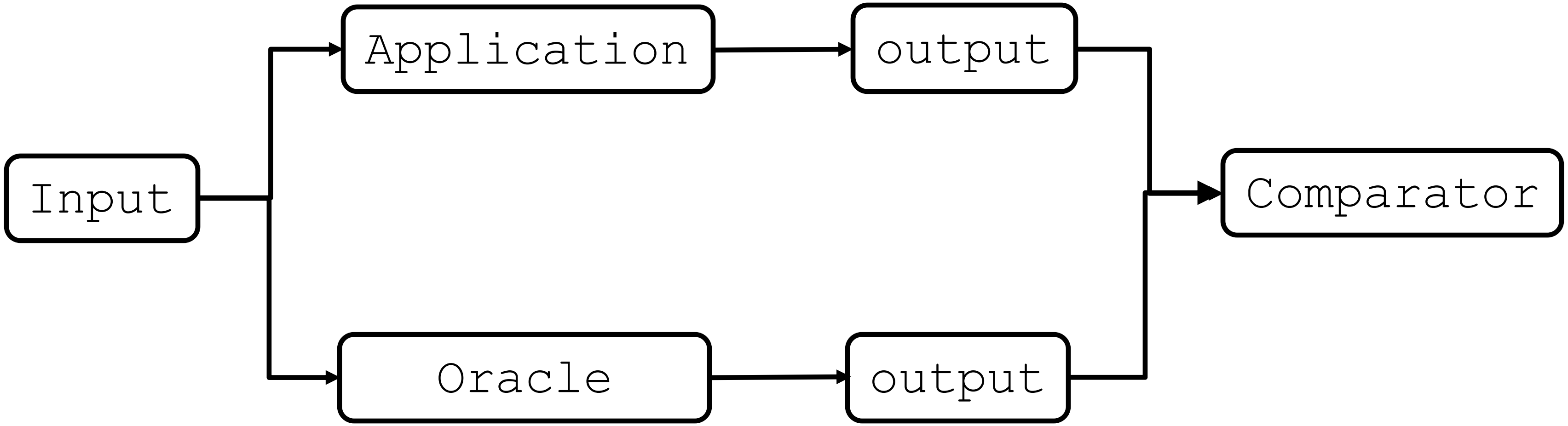
Code coverage

- **Function coverage** – Has each function been called?
- **Statement coverage** – Has each statement been executed?
- **Branch coverage** – Has each branch of each control structure (such as in *if* and *case* statements) been executed?
- **Condition coverage** (or predicate coverage) – Has each Boolean sub-expression evaluated both to true and false?
- Many more

Coverage Based Randomized Testing



Differential Testing



Property Based Testing

- Setting Up Junit-QuickCheck

- Maven

```
<dependency>
<groupId>com.pholser</groupId>
<artifactId>junit-quickcheck-core</artifactId>
<version>0.7</version>
</dependency>
```

- Eclipse**:**
  - Add the jar files

Let’s Test Our Property

```
@RunWith(JUnitQuickcheck.class)
public class PBT {
 @Property (trials = 1000)
 public void testList(List<String> l1) {
   List<String> l2 = l1.stream().collect(Collectors.toList());
   Collections.reverse(l2);
   Collections.reverse(l2);
   assertEquals(l1, l2);
 }
}
```

Test 1000 times

Generates a random string list

...and tests the property

Buggy Reverse

```
Reverse(List<?> l){ return l} //returns the same list
```

The property did not catch the bug!

```
reverse((reverse (l))) == l
```

A simple unit test would catch the bug

assertEquals (**reverse ([1,2,3]), [3,2,1])**

Another Property

```
testRev (List<Integer>l1, Integer x, List<Integer l2){
   assertEquals(
      rev (l1 ++ [x] ++ l2) , rev l2 ++ [x] ++ rev l1
   )
}
```

```
rev [1,2]++[3]@[4;5] = rev [4,5] ++ rev [3] ++ rev [1;2]
```

Junit-QuickCheck

- **junit-quickcheck: Property-based testinga, JUnit-style**

  `github:` [**`https://github.com/pholser/junit-quickcheck`**](https://github.com/pholser/junit-quickcheck)
- Documentation:
  - https://pholser.github.io/junit-quickcheck/site/1.0/

- Generator: random generators
- Shrink: Producing “smaller” values
- Seed: source of randomness

Demo

https://github.com/anwarmamat/cmsc330/tree/master/java/junit_quickcheck