EN.601.422 / EN.601.622
Software Testing & Debugging

The Goal of Software Testing

▶ In the last class we discussed:
  ❖ Why software testing matters
  ❖ "the goal of software testing"
    ● show presence of bugs, not their absence
  ❖ Fault vs error vs failure
  ❖ Testing is a destructive process!
  ❖ Verification vs. Validation

Plan for Today

▶ The **V** model, Test early rather than late

▶ RIPR Fault/Failure Model

▶ Oracle problem

▶ Warming up with JUnit

▶ Review Few Important Testing principles

▶ Exercise

The V Model
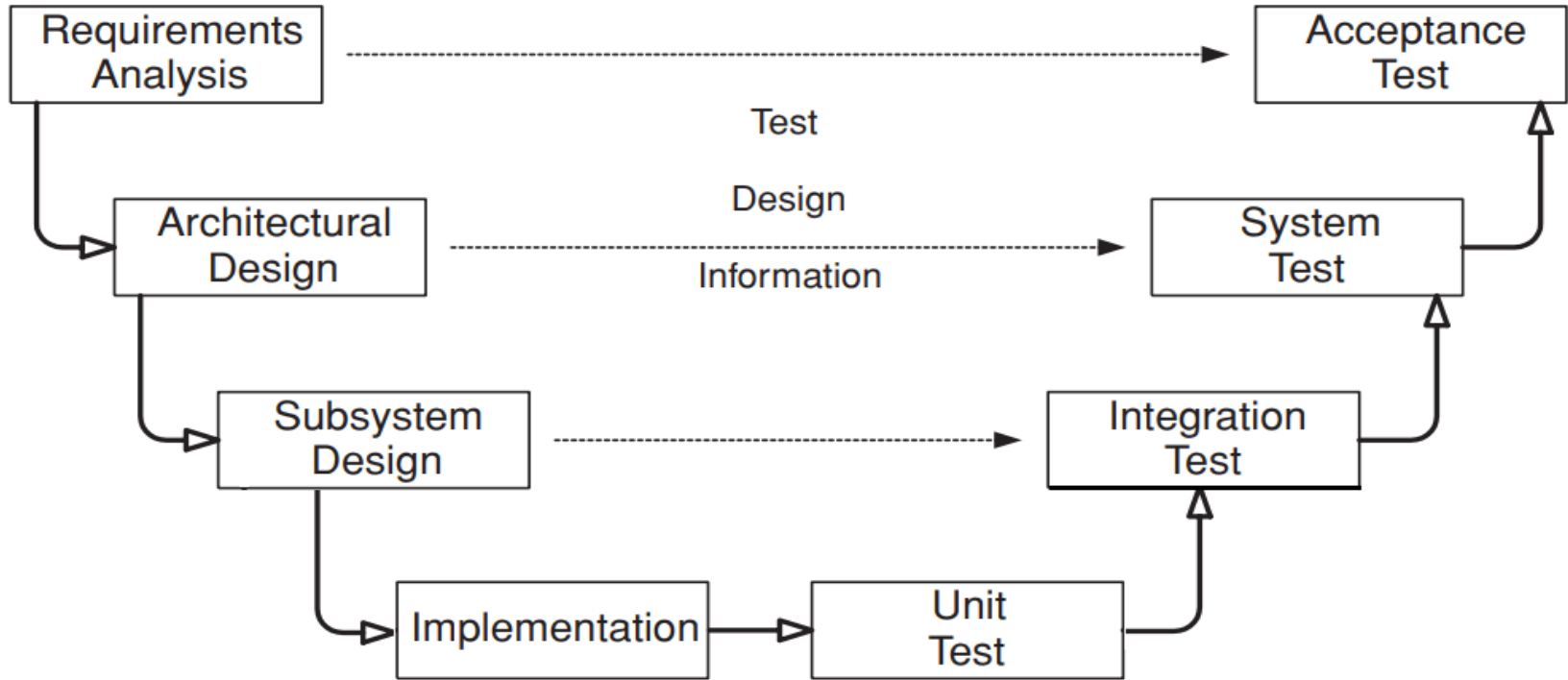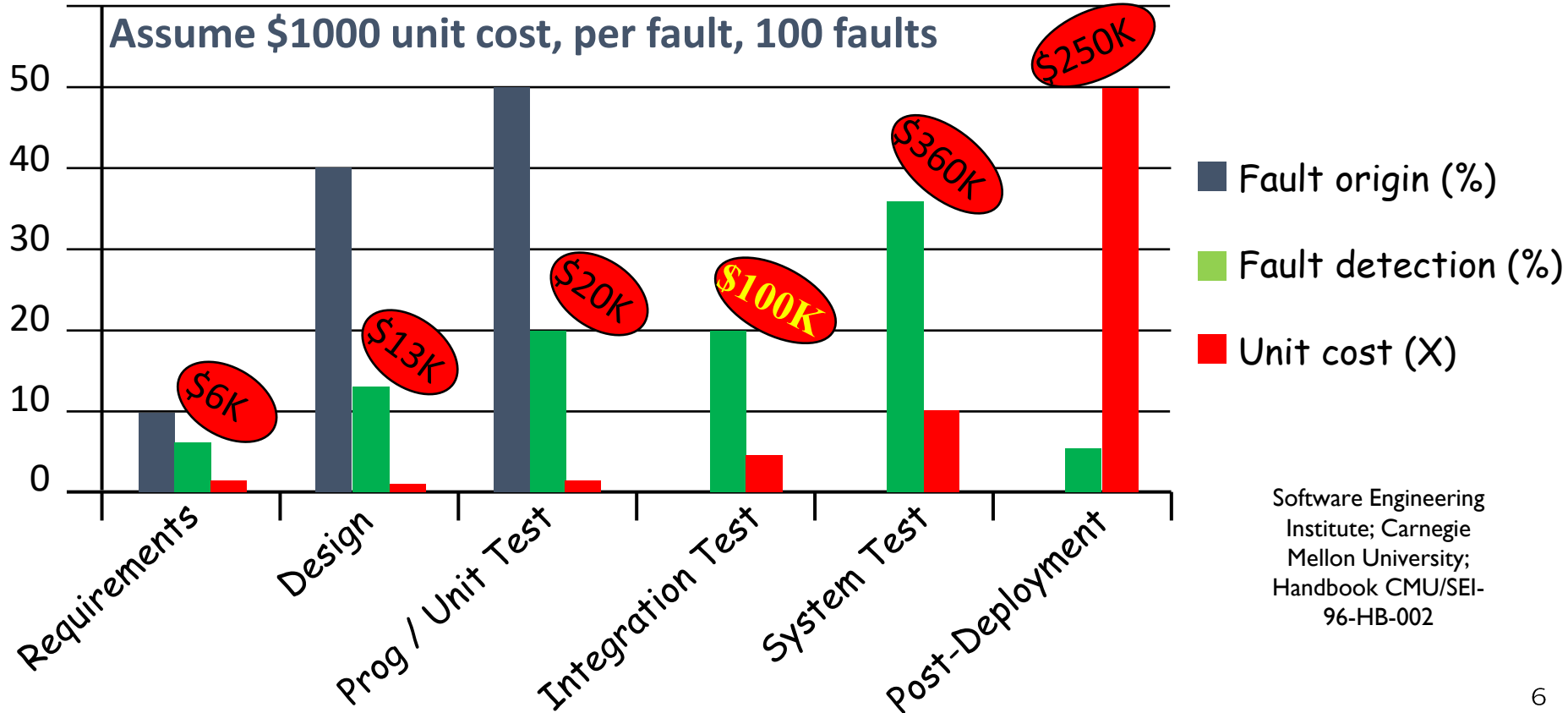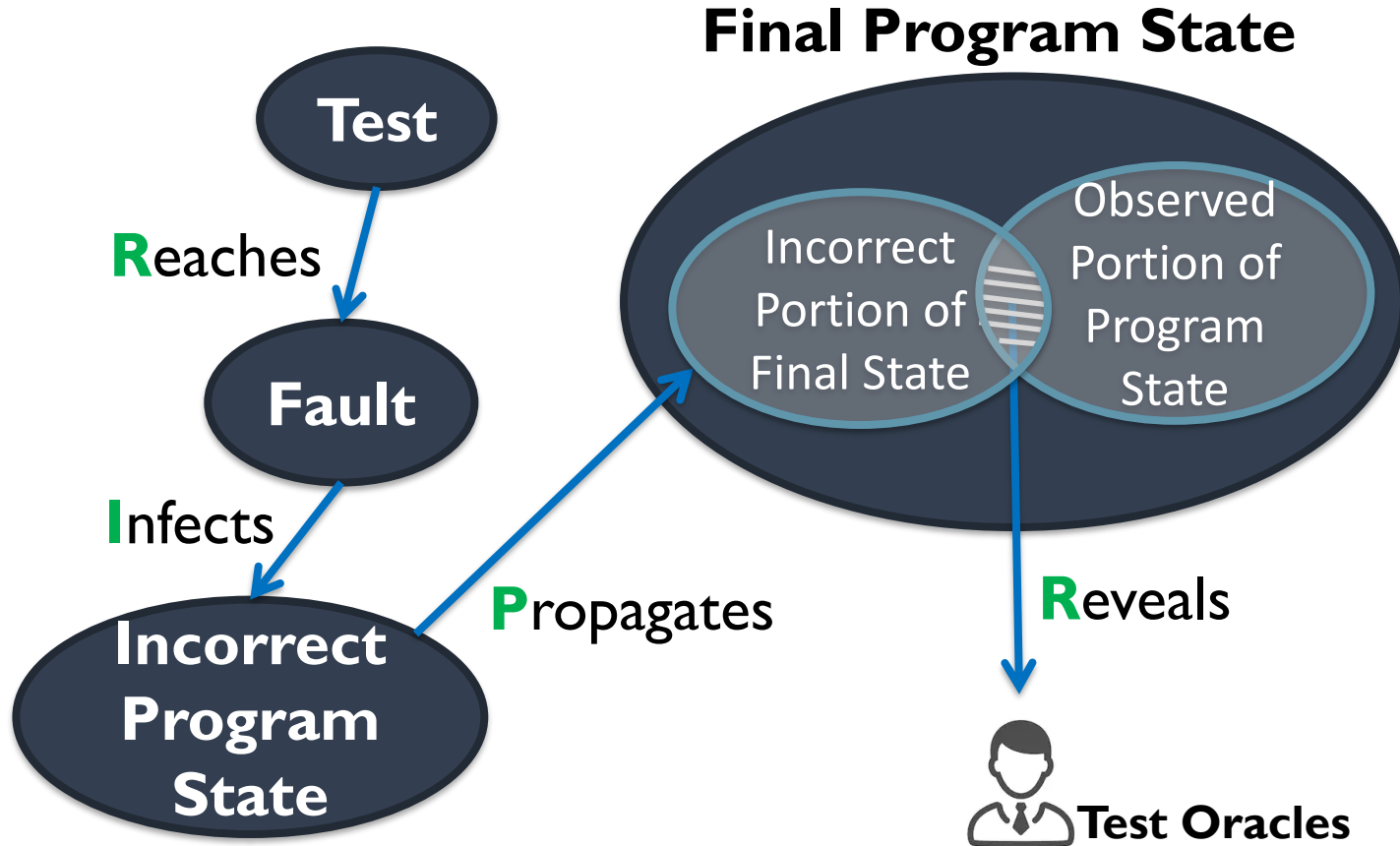
Testing Levels

► **Acceptance Testing**: assess software with respect to user's needs
  ❖ Alpha & Beta Testing

► **System Testing**: assess software with respect to architectural design and overall expected behavior

► **Integration Testing**: assess software with respect to sub-system design

► **Unit Testing**: assess software with respect to implementation of isolated modules/units

Testing Early vs. Testing Late

Assume $1000 unit cost, per fault, 100 faults

Legend:
- Fault origin (%)
- Fault detection (%)
- Unit cost (X)

Categories: Requirements, Design, Prog / Unit Test, Integration Test, System Test, Post-Deployment

Callouts: $6K, $13K, $20K, $100K, $360K, $250K

Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002

RIPR or Fault/Failure Model

Final Program State

Test

Reaches

Fault

Infects

Incorrect Program State

Propagates

Incorrect Portion of Final State

Observed Portion of Program State

Reveals

Test Oracles

8

RIPR Model

▶ **Reachability**: The location or locations in the program that contain the fault must be reached

▶ **Infection**: The state of the program must be incorrect

▶ **Propagation**: The infected state must cause some output or final state of the program to be incorrect

▶ **Reveal**: The tester must observe part of the incorrect portion of the program state

Test Automation

▶ **Two Types of Testing**:
- ❖ Manual
- ❖ Automated

▶ **Test Automation**: automation of testing-related activities
- ❖ **Generation**: generate test cases automatically
- ❖ **Execution**: run tests on the software under test (SUT)
- ❖ **Evaluation**: evaluate test results i.e., does the test case pass or fail

Test Evaluation

▶ **Test Oracle**: a mechanism for determining whether a test has passed or failed. An oracle can be:
- ❖ Expected output value
- ❖ A program
- ❖ Documentation that gives specific correct outputs for specific given inputs
- ❖ A (human) domain expert who can tell whether test output is correct or not
- ❖ Any other way or combination of the above that can tell that output is correct or not

Examples

► Unit Testing:
  ❖ E.g., **assertEquals(4, sum(2, 2))**: 4 is the oracle and hard coded!
  ❖ Is the above oracle complete?

Examples

► Unit Testing:

❖ E.g., **assertEquals(4, sum(2, 2))**: 4 is the oracle and hard coded!

❖ Is the above oracle complete?

```
public class MyClass {
    int c;

    public int sum(int a, int b) {
        c  = 10;
        return a + b;
    }
    …
}
```

More Examples (Other Types of Testing)

► System Testing:
  ❖ E.g., Testing Google search engine with a query
  ❖ What set of results should it return exactly for the query?

► Security Testing:
  ❖ E.g., a test case that simulates a sql injection attack
  ❖ How and what test oracle would you write? i.e., what is exactly the expected behavior?

► Usability testing:
  ❖ E.g., testing the Graphical User Interface of a web app for user friendliness
  ❖ A test case would be accomplishing a task using the GUI. Was it user friendly enough?

Test Oracles

▶ A *complete* Oracle would be based on the entire final state after running a test case
  ❖ Impractical/impossible

▶ Weak (partial) oracles:
  ❖ Usually, good enough in practice
  ❖ Examples:
    ● check for expected output
    ● check for software crashes
    ● Etc.

Test Evaluation

► One of the most challenging problems in software testing

► Much harder than it might seem; it might not be straightforward what the correct/expected output is/should be

► Requires knowledge of domain, user interfaces, psychology etc.

► This is known as:

The Oracle Problem!

xUnit

► xUnit is the collective name for several unit testing frameworks
  ❖ SUnit was the first one for smalltalk
  ❖ Ported to Java (JUnit) by Kent Beck and Erich Gamma which gained wide popularity

► Available as a library that can be added to your Java project

Follow this JUnit Code Skeleton (for now)

```java
public class ClassNameTest {

    @Test
    public void testFunctionNameTestDescription () {
        // make use of a variety of assert statements provided
    }
    // more test functions
}
```

Testing Principle 1

A necessary part of any test case is a definition of the expected output/behavior

Testing Principle 2

(Ideally) a programmer should avoid attempting to test her own program

Debugging is best done by the original developer though

Testing Principle 3

► Faults are not uniformly distributed
  ❖ The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.

Probability of the existence of additional errors

Number of errors found already
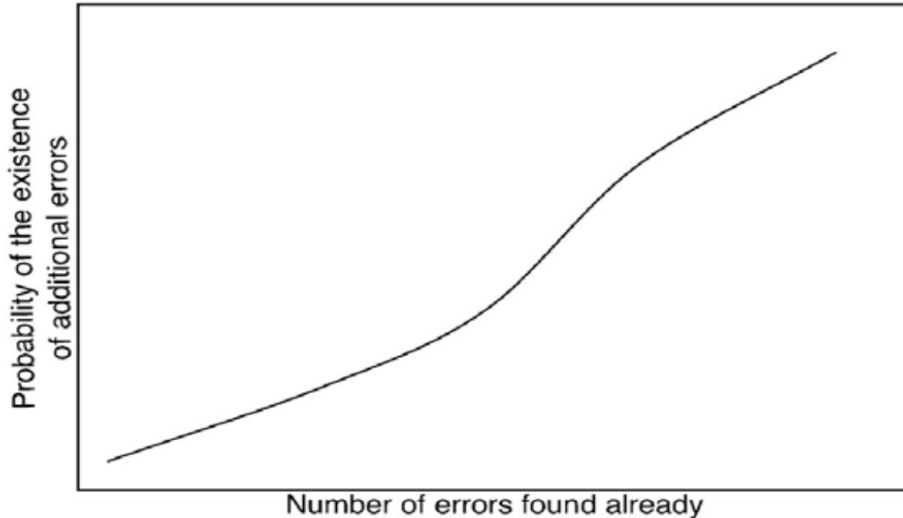
Testing Principle 4

▶ Examining a program to see if it does not do what it is supposed to do is only half the battle; the other half is seeing whether the program does it is not supposed to do