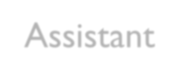
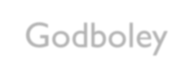
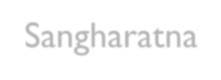
Introduction to Testing

Dr. Sangharatna Godboley, 

Assistant Professor, 

Department Of CSE, NIT Warangal.Slides Credits: Dr. Durga Prasad Mohapatra, Professor, Department Of CSE, NIT, Rourkela.

1

Defect Reduction Techniques ⚫Review 

⚫Testing

⚫Formal verification

⚫Development process

⚫Systematic methodologies

2

Why Test?



• Ariane 5 rocket self-destructed 37 seconds after launch • Reason: A control software bug that went undetected – Conversion from 64-bit floating point to 16-bit signed integer value had caused an exception

• The floating point number was larger than 32767

• Efficiency considerations had led to the disabling of the exception handler. • Total Cost: over $1 billion

3

How Do You Test a Program? ⚫ Input test data to the program. ⚫ Observe the output: 

◦ Check if the program behaved as expected.

4

How Do You Test a Program? 5

How Do You Test a Program? ⚫ If the program does not behave as expected: 

◦ Note the conditions under which it failed.

◦ Later debug and correct.

6

What’s So Hard About Testing ? ⚫ Consider **int proc1(int x, int y)** ⚫ Assuming a 64 bit computer

◦ Input space = 2128

⚫Assuming it takes 10secs to key-in an integer pair ◦It would take about a billion years to enter all  possible values!

◦Automatic testing has its own problems!

7

Testing Facts 

⚫Consumes largest effort among all phases ◦ Largest manpower among all other development roles 

◦Implies more job opportunities

⚫About 50% development effort ◦ But 10% of development time?

◦ How?

8

Testing Facts 

⚫Testing is getting more complex and sophisticated every year.

◦ Larger and more complex programs ◦Newer programming paradigms

9

Overview of Testing Activities ⚫Test Suite Design 

⚫Run test cases and observe results to detect failures.

⚫Debug to locate errors

⚫Correct errors.

10

Error, Faults, and Failures ⚫A failure is a manifestation of an error (also defect or bug). 

◦ Mere presence of an error may not lead to a failure.

11

Pesticide Effect 

⚫ Errors that escape a fault detection technique: ◦ Can not be detected by further applications of that technique.

12

Pesticide Effect 

⚫Assume we use 4 fault detection techniques and 1000 bugs:

◦ Each detects only 70% bugs

◦ How many bugs would remain ◦ 1000\*(0.3)4=81 bugs

13

Fault Model 

⚫Types of faults possible in a program.

⚫ Some types can be ruled out ◦ Concurrency related-problems in a sequential program

14

Fault Model of an OO Program

**OO Faults**

**Structural**

**FaultsAlgorithmic  Faults**

**Incorrect**

**Procedural**

**FaultsTraceability**

**FaultsOO**

**Faults**

**ResultInadequate  Performance**

15

Hardware Fault-Model 

⚫ Simple:

◦ Stuck-at 0

◦ Stuck-at 1

◦ Open circuit

◦ Short circuit

⚫ Simple ways to test the presence of each ⚫ Hardware testing is fault-based testing

16

Software Testing 

⚫ Each test case typically tries to establish correct working of some functionality ◦ Executes (covers) some program elements ◦ For restricted types of faults, fault-based testing exists.

17

Test Cases and Test Suites 

⚫Test a software using a set of carefully designed test cases: ◦The set of all test cases is called the test suite

18

Test Cases and Test Suites ⚫A test case is a triplet [I,S,O] 

◦I is the data to be input to the system,

◦ S is the state of the system at which the data will be input,

◦ O is the expected output of the system.

19

Verification versus Validation 

⚫ Verification is the process of

determining:

◦ Whether output of one phase of development conforms to its previous phase.

⚫ Validation is the process of determining: ◦ Whether a fully developed system conforms to its SRS document.

20

Verification versus Validation ⚫Verification is concerned with phase containment of errors, ◦Whereas the aim of validation is that the final product be error free. 

21

Design of Test Cases 

⚫ Exhaustive testing of any non-trivial system is impractical:

◦ Input data domain is extremely large. ⚫Design an optimal test suite: ◦ Of reasonable size and

◦ Uncovers as many errors as possible. 22

Design of Test Cases 

⚫ If test cases are selected randomly:

◦ Many test cases would not contribute to the significance of the test suite,

◦ Would not detect errors not already being detected by other test cases in the suite.

⚫ Number of test cases in a randomly selected test suite:

◦ Not an indication of effectiveness of testing.

23

Design of Test Cases 

⚫ Testing a system using a large number of randomly selected test cases:

◦ Does not mean that many errors in the system will be uncovered.

⚫ Consider following example:

◦ Find the maximum of two integers x and y.

24

Design of Test Cases 

⚫ The code has a simple programming error: ⚫ If (x>y) max = x;

else max = x;

⚫ Test suite {(x=3,y=2);(x=2,y=3)} can detect the error,

⚫ A larger test suite {(x=3,y=2);(x=4,y=3); (x=5,y=1)} does not detect the error.

25

Design of Test Cases 

⚫ Systematic approaches are required to design an optimal test suite:

◦ Each test case in the suite should detect different errors.

26

Design of Test Cases 

⚫There are essentially three main approaches to design test cases:

◦Black-box approach

◦White-box (or glass-box)

approach

◦Grey-box testing

27

Black-Box Testing 

⚫ Test cases are designed using only functional specification of the software: ◦ Without any knowledge of the internal structure of the software.

⚫ For this reason, black-box testing is also known as functional testing.

28

White-box Testing 

⚫Designing white-box test cases: ◦ Requires knowledge about the internal structure of software.

◦White-box testing is also called structural testing.

◦ In this unit we will not study white-box testing.

29

White-Box Testing 

⚫ There exist several popular white-box testing methodologies:

◦ Statement coverage

◦ Branch coverage

◦ Path coverage

◦ Condition coverage

◦ MC/DC coverage

◦ Mutation testing

◦ Data flow-based testing

30

Why Both BB and WB Testing?

Black-box 

⚫ Impossible to write a test case for every possible set of inputs and outputs

⚫ Some code parts may not be reachable

⚫ Does not tell if extra functionality has been implemented.

White-box

⚫ Does not address the question of whether or not a program matches the specification

⚫ Does not tell you if all of the functionality has been implemented

⚫ Does not discover missing program logic

31

Coverage-Based Testing Versus 

Fault-Based Testing 

⚫ Idea behind coverage-based testing: ◦ Design test cases so that certain program elements are executed (or covered).

◦ Example: statement coverage, path coverage, etc.

⚫ Idea behind fault-based testing:

◦ Design test cases that focus on discovering certain types of faults.

◦ Example: Mutation testing.

32



33

**Thank You**