



Software Testing:
Review


 University of Minnesota
 Software Engineering Center

1

Verification vs. Validation



 University of Minnesota
 Software Engineering Center

2

http://www.umsec.umn.edu

- Verification:
 - ◆ ?????
- Validation:
 - ◆ ?????

**Verification vs. Validation:
Right Definition**


 University of Minnesota
 Software Engineering Center

3

http://www.umsec.umn.edu


- Verification:
 - ◆ “are we building the product right?”
 - ◆ The software should conform to its specification
- Validation:
 - ◆ “are we building the right product?”
 - ◆ The software should do what the user really requires

**Assuring that a software system
meets a user’s needs**

Dynamic and Static Verification

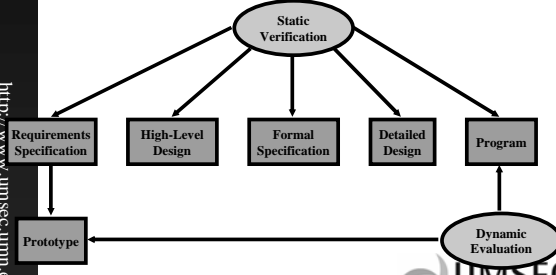
- Dynamic V & V
 - ◆ Concerned with exercising and observing product behavior
 - ◆ Testing
- Static V & V
 - ◆ Concerned with analysis of the static system representation to discover problems
 - ◆ Proofs
 - ◆ Inspections

<http://www.umsec.umn.edu>




4

Static and Dynamic V&V



<http://www.umsec.umn.edu>




5

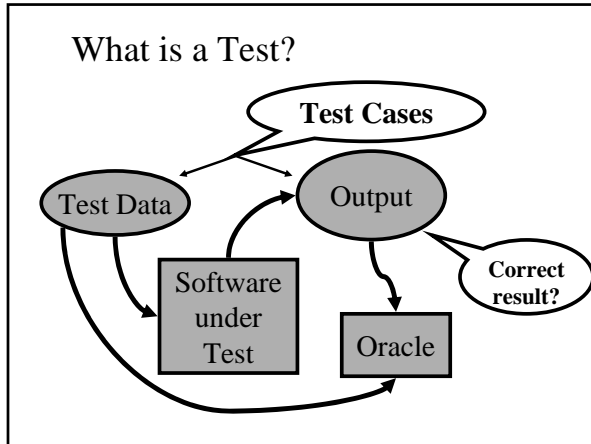
Definitions of Testing

- The process of executing a program (or part of a program) with the intention of finding errors (Myers, via Humphrey)
- The purpose of testing is to find errors
 - ◆ Testing is the process of trying to discover every conceivable fault or weakness in a work product (Myers, via Kit)
- The process of searching for errors (Kaner)

<http://www.umsec.umn.edu>



6



Test Data and Test Cases

- Test data
 - ◆ Inputs which have been devised to test the system
- Test cases
 - ◆ Inputs to test the system and the predicted outputs from these inputs if the system operates according to its specification

<http://www.umsec.umn.edu>

8

Bugs? What is That?

- Failure
 - ◆ An execution that yields an erroneous result
- Fault
 - ◆ The source of the failure
- Error
 - ◆ The mistake that led to the fault being introduced in the code

<http://www.umsec.umn.edu>


9

Axiom of Testing

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

Dijkstra, 1969

<http://www.umsec.umn.edu>




10

Black and White Box

- Black box testing:
 - ◆ Designed without knowledge of the program’s internal structure and design
 - ◆ Based on functional requirements
- White box testing:
 - ◆ Examines the internal design of the program
 - ◆ Requires detailed knowledge of its structure

<http://www.umsec.umn.edu>




11

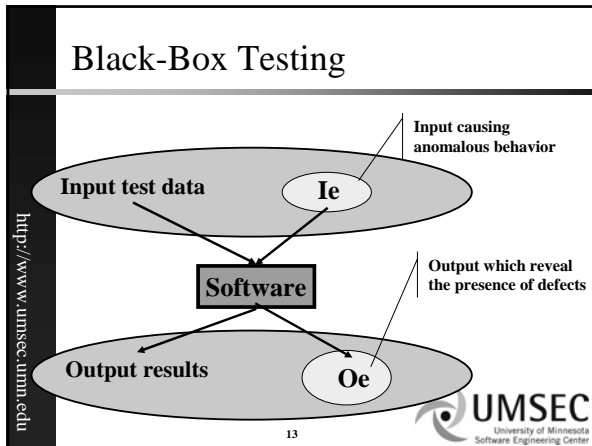
Black-Box Testing

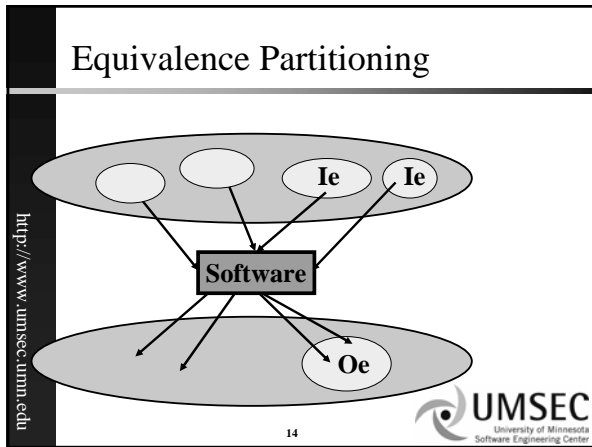
- Approach to testing where the program is considered as a “black-box”
- The program test cases are based on the system specification
- Test planning can begin early in the software process

<http://www.umsec.umn.edu>



12





- ### Structural Coverage Testing
- (In)adequacy criteria
 - ◆ If significant parts of program structure are not tested, testing is surely inadequate
 - Control flow coverage criteria
 - ◆ Statement (node, basic block) coverage
 - ◆ Branch (edge) coverage
 - ◆ Condition coverage
 - ◆ Path coverage
 - ◆ Data flow (syntactic dependency) coverage
 - Attempted compromise between the impossible and the inadequate
- A vertical URL "http://www.umsec.umn.edu" is on the left, and the "UMSEC University of Minnesota Software Engineering Center" logo is at the bottom right. The number "15" is centered at the bottom.

Statement Coverage

```

int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
    if (A[i]<0)
      A[i] = - A[i];
    i++;
  }
  return(1);
}
    
```

One test datum (N=1, A[0]=-7, X=9) is enough to guarantee statement coverage of function select
 Faults in handling positive values of A[i] would not be revealed

UMSEC
University of Minnesota
Software Engineering Center

16

Branch Coverage

```

int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
    if (A[i]<0)
      A[i] = - A[i];
    i++;
  }
  return(1);
}
    
```

We must add a test datum (N=1, A[0]=7, X=9) to cover branch False of the if statement. Faults in handling positive values of A[i] would be revealed. Faults in exiting the loop with condition A[i] < X would not be revealed

UMSEC
University of Minnesota
Software Engineering Center

17

Path Coverage

```

int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
    if (A[i]<0)
      A[i] = - A[i];
    i++;
  }
  return(1);
}
    
```

The loop must be iterated given number of times.
PROBLEM: uncontrolled growth of test sets. We need to select a significant subset of test cases.

UMSEC
University of Minnesota
Software Engineering Center

18

Condition Coverage

```

int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
    if (A[i]<0)
      A[i] = - A[i];
    i++;
  }
  return(1);
}
    
```

Both conditions (i<N), (A[i]<X) must be false and true for different tests. In this case, we must add tests that cause the while loop to exit for a value greater than X. Faults that arise after several iterations of the loop would not be revealed.

UMSEC
University of Minnesota
Software Engineering Center

19

Basic Condition Coverage

- Make each condition both True and False

Test Case	Cond 1	Cond 2
1	True	False
2	False	True

UMSEC
University of Minnesota
Software Engineering Center

20

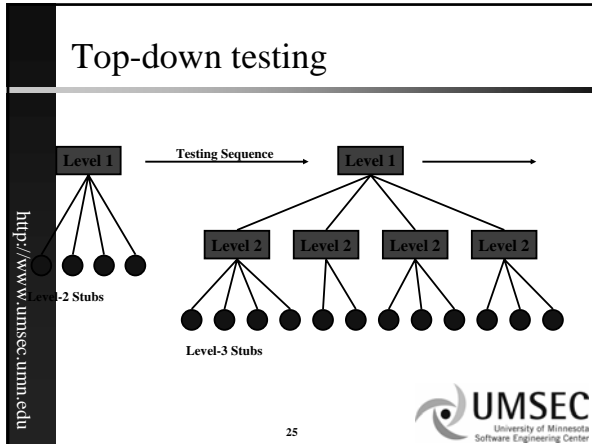
Compound Condition Coverage

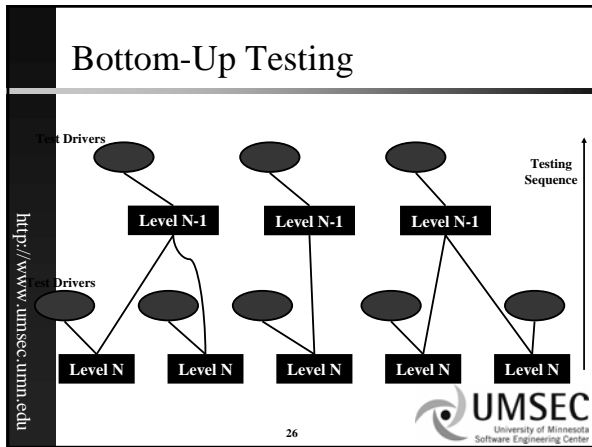
- Evaluate every combination of the conditions

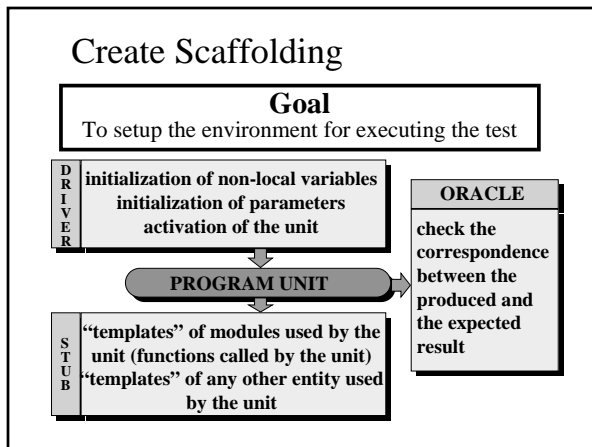
Test Case	Cond 1	Cond 2
1	True	True
2	True	False
3	False	True
4	False	False

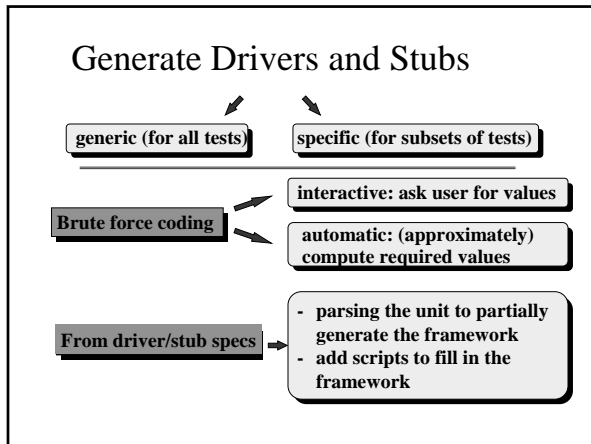
UMSEC
University of Minnesota
Software Engineering Center

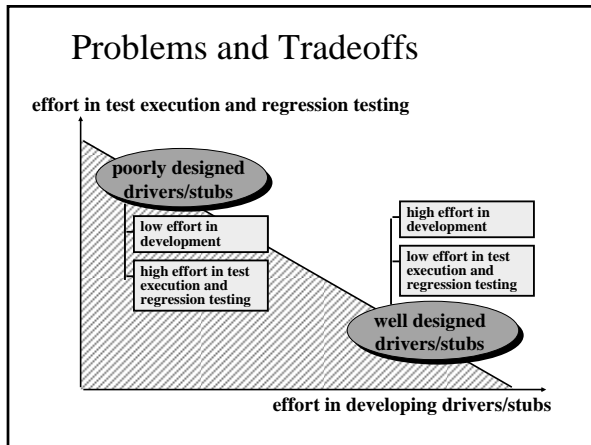
21











Who Should Test?

• **Developer**

- ◆ Understands the system
- ◆ But, will test gently
- ◆ And, is driven by deadlines

• **Independent tester**

- ◆ Must learn system
- ◆ But, will attempt to break it
- ◆ And, is driven by "quality"


<http://www.umsec.umn.edu>

<http://www.umsec.umn.edu>

Axioms of Testing

- As the number of detected defects in a piece of software increases, the probability of the existence of more undetected defects also increases
- Assign your best programmers to testing
- Exhaustive testing is impossible

31



<http://www.umsec.umn.edu>

Axioms of Testing

- You cannot test a program completely
- Even if you do find the last bug, you'll never know it
- It takes more time than you have to test less than you'd like
- You will run out of time before you run out of test cases

32

