

Chapter 18

■ Testing Conventional Applications

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e
by Roger S. Pressman

Slides copyright © 1996, 2001, 2005, 2009 by Roger S. Pressman

For non-profit educational use only

May be reproduced ONLY for student use at the university level when used in conjunction with *Software Engineering: A Practitioner's Approach, 7/e*. Any other reproduction or use is prohibited without the express written permission of the author.

All copyright information MUST appear if these slides are posted on a website for student use.

Testability

- **Operability**—it operates cleanly
- **Observability**—the results of each test case are readily observed
- **Controllability**—the degree to which testing can be automated and optimized
- **Decomposability**—testing can be targeted
- **Simplicity**—reduce complex architecture and logic to simplify tests
- **Stability**—few changes are requested during testing
- **Understandability**—of the design

What is a “Good” Test?

- A good test has a high probability of finding an error
- A good test is not redundant.
- A good test should be “best of breed”
- A good test should be neither too simple nor too complex

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

3

Internal and External Views

- Any engineered product (and most other things) can be tested in one of two ways:
 - Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function;
 - Knowing the internal workings of a product, tests can be conducted to ensure that "all gears mesh," that is, internal operations are performed according to specifications and all internal components have been adequately exercised.

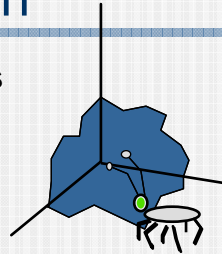
These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

4

Test Case Design

"Bugs lurk in corners
and congregate at
boundaries ..."

Boris Beizer



OBJECTIVE to uncover errors

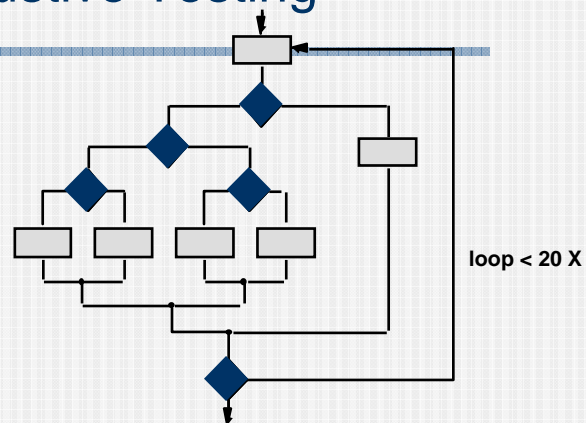
CRITERIA in a complete manner

CONSTRAINT with a minimum of effort and time

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

5

Exhaustive Testing

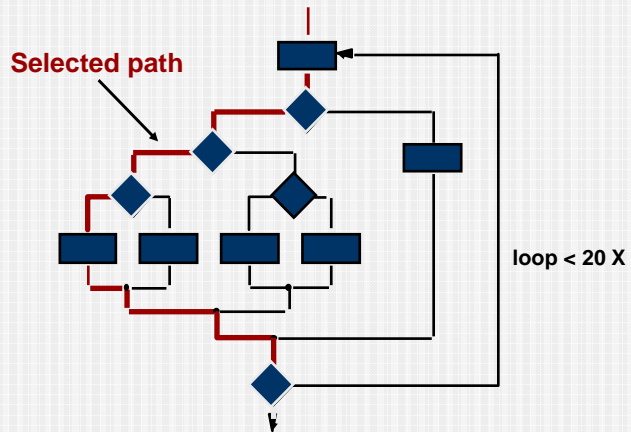


**There are 10^{14} possible paths! If we execute one
test per millisecond, it would take 3,170 years to
test this program!!**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

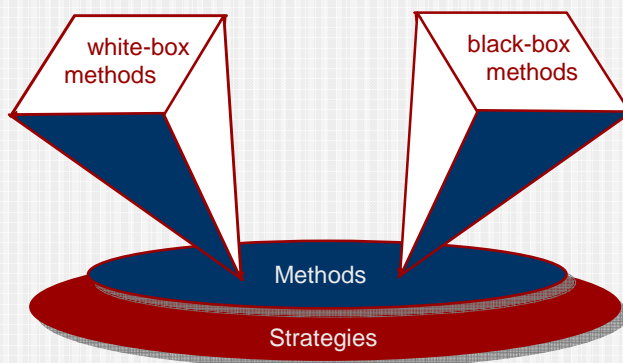
6

Selective Testing



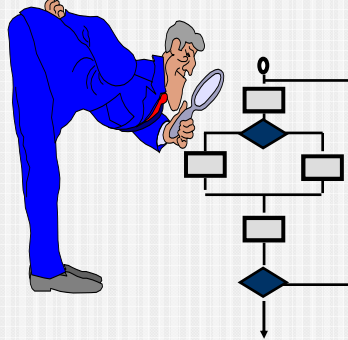
These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Software Testing



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

White-Box Testing



... our goal is to ensure that all statements and conditions have been executed at least once ...

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

9

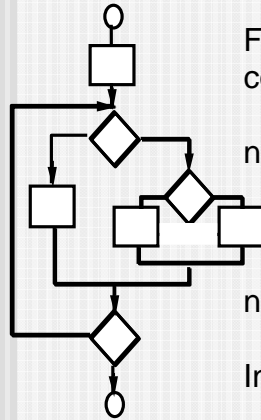
Why Cover?

- **logic errors and incorrect assumptions are inversely proportional to a path's execution probability**
- **we often believe that a path is not likely to be executed; in fact, reality is often counter intuitive**
- **typographical errors are random; it's likely that untested paths will contain some**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

10

Basis Path Testing



First, we compute the cyclomatic complexity:

number of simple decisions + 1

or

number of enclosed areas + 1

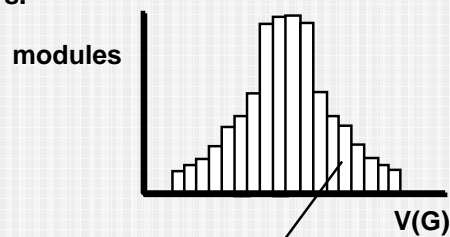
In this case, $V(G) = 4$

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

11

Cyclomatic Complexity

A number of industry studies have indicated that the higher $V(G)$, the higher the probability of errors.

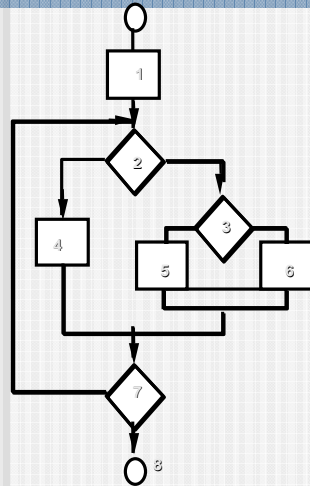


modules in this range are more error prone

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

12

Basis Path Testing



Next, we derive the independent paths:

Since $V(G) = 4$, there are four paths

Path 1: 1,2,3,6,7,8

Path 2: 1,2,3,5,7,8

Path 3: 1,2,4,7,8

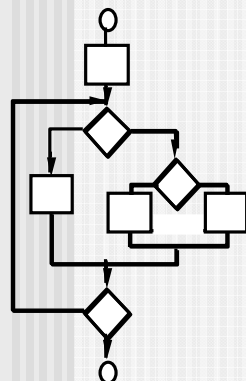
Path 4: 1,2,4,7,2,4,...7,8

Finally, we derive test cases to exercise these paths.

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

13

Basis Path Testing Notes



- you don't need a flow chart, but the picture will help when you trace program paths
- count each simple logical test, compound tests count as 2 or more
- basis path testing should be applied to critical modules

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

14

Deriving Test Cases

- *Summarizing:*
 - Using the design or code as a foundation, draw a corresponding flow graph.
 - Determine the cyclomatic complexity of the resultant flow graph.
 - Determine a basis set of linearly independent paths.
 - Prepare test cases that will force execution of each path in the basis set.

Graph Matrices

- A graph matrix is a square matrix whose size (i.e., number of rows and columns) is equal to the number of nodes on a flow graph
- Each row and column corresponds to an identified node, and matrix entries correspond to connections (an edge) between nodes.
- By adding a *link weight* to each matrix entry, the graph matrix can become a powerful tool for evaluating program control structure during testing

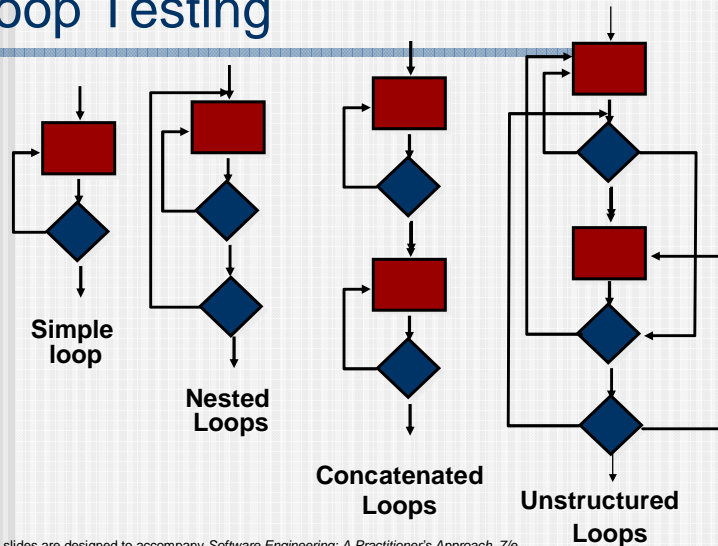
Control Structure Testing

- **Condition testing** — a test case design method that exercises the logical conditions contained in a program module
- **Data flow testing** — selects test paths of a program according to the locations of definitions and uses of variables in the program

Data Flow Testing

- The data flow testing method [Fra93] selects test paths of a program according to the locations of definitions and uses of variables in the program.
 - Assume that each statement in a program is assigned a unique statement number and that each function does not modify its parameters or global variables. For a statement with S as its statement number
 - $DEF(S) = \{X \mid \text{statement } S \text{ contains a definition of } X\}$
 - $USE(S) = \{X \mid \text{statement } S \text{ contains a use of } X\}$
 - A **definition-use (DU) chain** of variable X is of the form $[X, S, S']$, where S and S' are statement numbers, X is in $DEF(S)$ and $USE(S')$, and the definition of X in statement S is live at statement S'

Loop Testing



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

19

Loop Testing: Simple Loops

Minimum conditions—Simple Loops

1. skip the loop entirely
2. only one pass through the loop
3. two passes through the loop
4. m passes through the loop $m < n$
5. $(n-1)$, n , and $(n+1)$ passes through the loop

where n is the maximum number of allowable passes

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

20

Loop Testing: Nested Loops

Nested Loops

Start at the innermost loop. Set all outer loops to their minimum iteration parameter values.

Test the min+1, typical, max-1 and max for the innermost loop, while holding the outer loops at their minimum values.

Move out one loop and set it up as in step 2, holding all other loops at typical values. Continue this step until the outermost loop has been tested.

Concatenated Loops

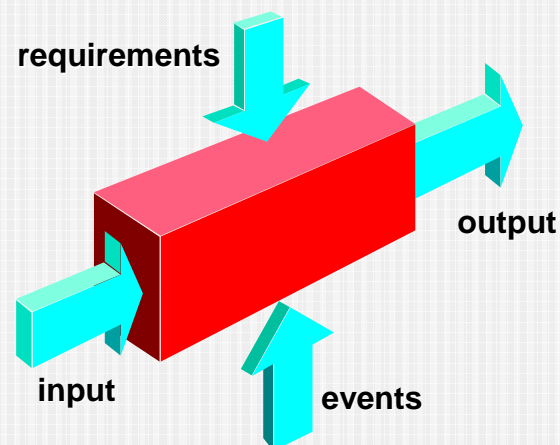
If the loops are independent of one another
then treat each as a simple loop
else* treat as nested loops
endif*

for example, the final loop counter value of loop 1 is used to initialize loop 2.

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

21

Black-Box Testing



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

22

Black-Box Testing

- How is functional validity tested?
- How is system behavior and performance tested?
- What classes of input will make good test cases?
- Is the system particularly sensitive to certain input values?
- How are the boundaries of a data class isolated?
- What data rates and data volume can the system tolerate?
- What effect will specific combinations of data have on system operation?

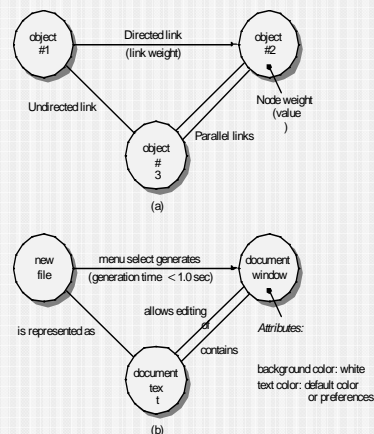
These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

23

Graph-Based Methods

To understand the objects that are modeled in software and the relationships that connect these objects

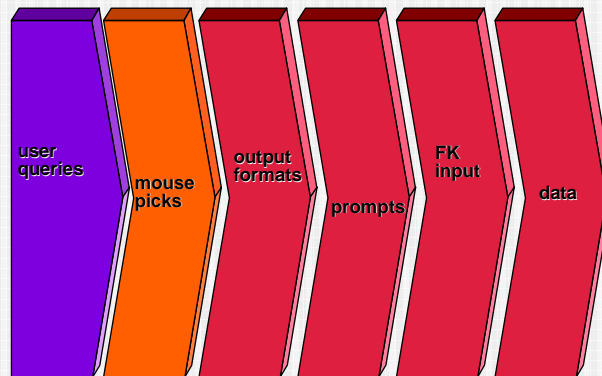
In this context, we consider the term "objects" in the broadest possible context. It encompasses data objects, traditional components (modules), and object-oriented elements of computer software.



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

24

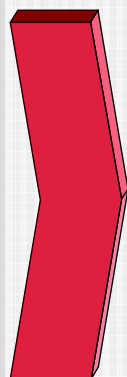
Equivalence Partitioning



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

25

Sample Equivalence Classes



Valid data

- user supplied commands
- responses to system prompts
- file names
- computational data
 - physical parameters
 - bounding values
 - initiation values
- output data formatting
- responses to error messages
- graphical data (e.g., mouse picks)

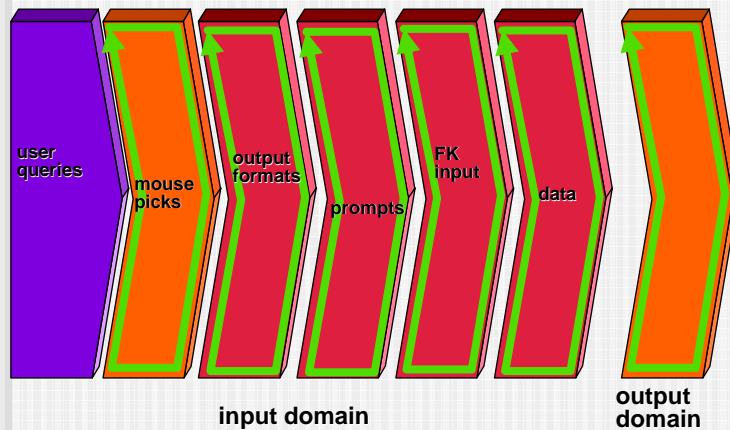
Invalid data

- data outside bounds of the program
- physically impossible data
- proper value supplied in wrong place

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

26

Boundary Value Analysis



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

27

Comparison Testing

- Used only in situations in which the reliability of software is absolutely critical (e.g., human-rated systems)
 - Separate software engineering teams develop independent versions of an application using the same specification
 - Each version can be tested with the same test data to ensure that all provide identical output
 - Then all versions are executed in parallel with real-time comparison of results to ensure consistency

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

28