Software Engineering: A Practitioner's Approach, 6/e

Chapter 13
Software Testing Strategies

copyright © 1996, 2001, 2005
R.S. Pressman & Associates, Inc.

For University Use Only
May be reproduced ONLY for student use at the university level
when used in conjunction with *Software Engineering: A Practitioner's Approach.*
Any other reproduction or use is expressly prohibited.

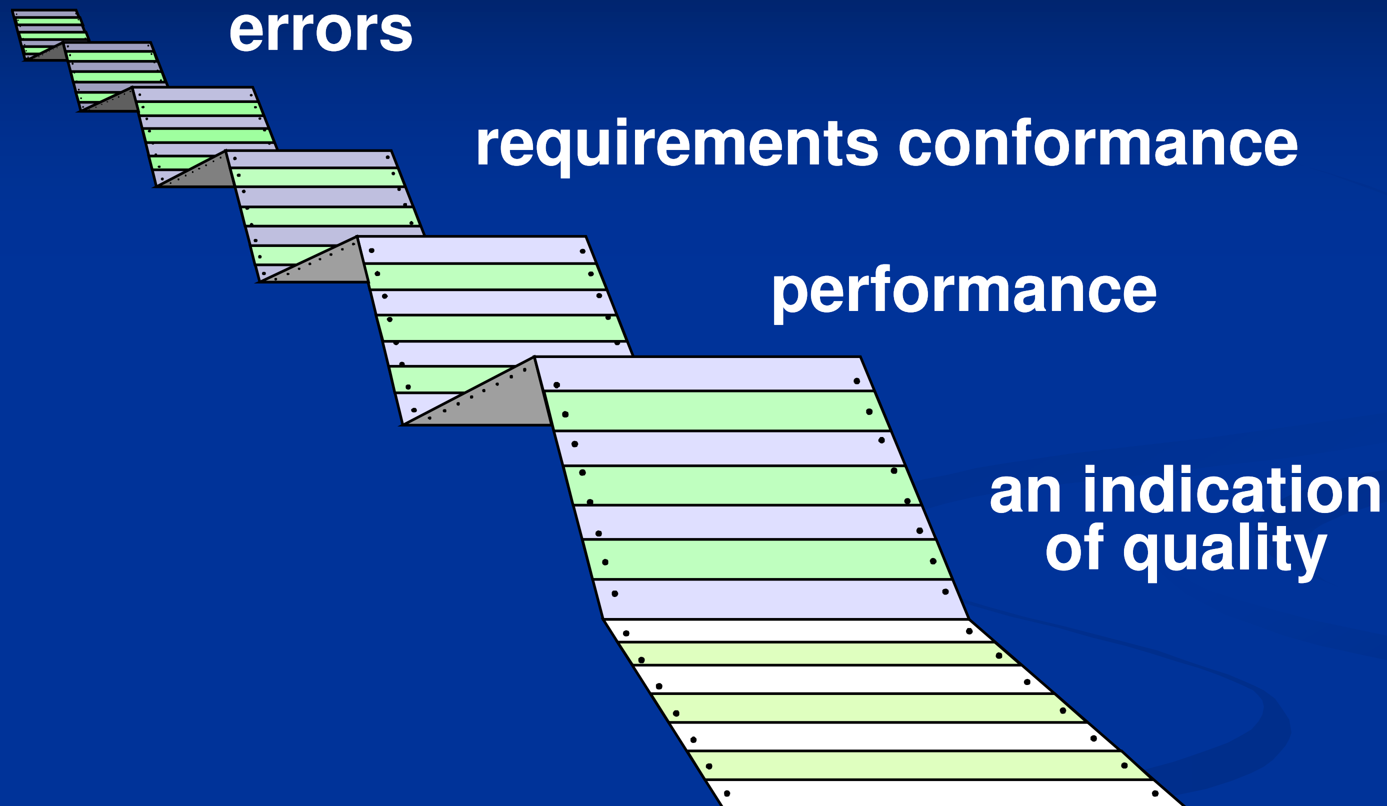
These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Software Testing

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.

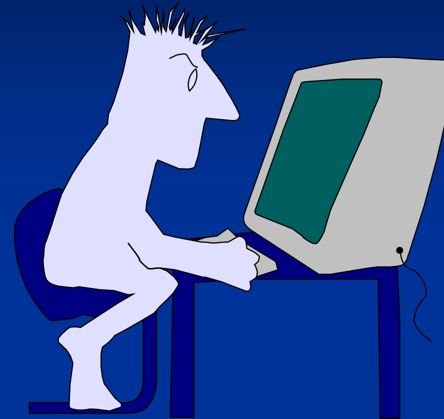
These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

What Testing Shows



These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Who Tests the Software?



developer

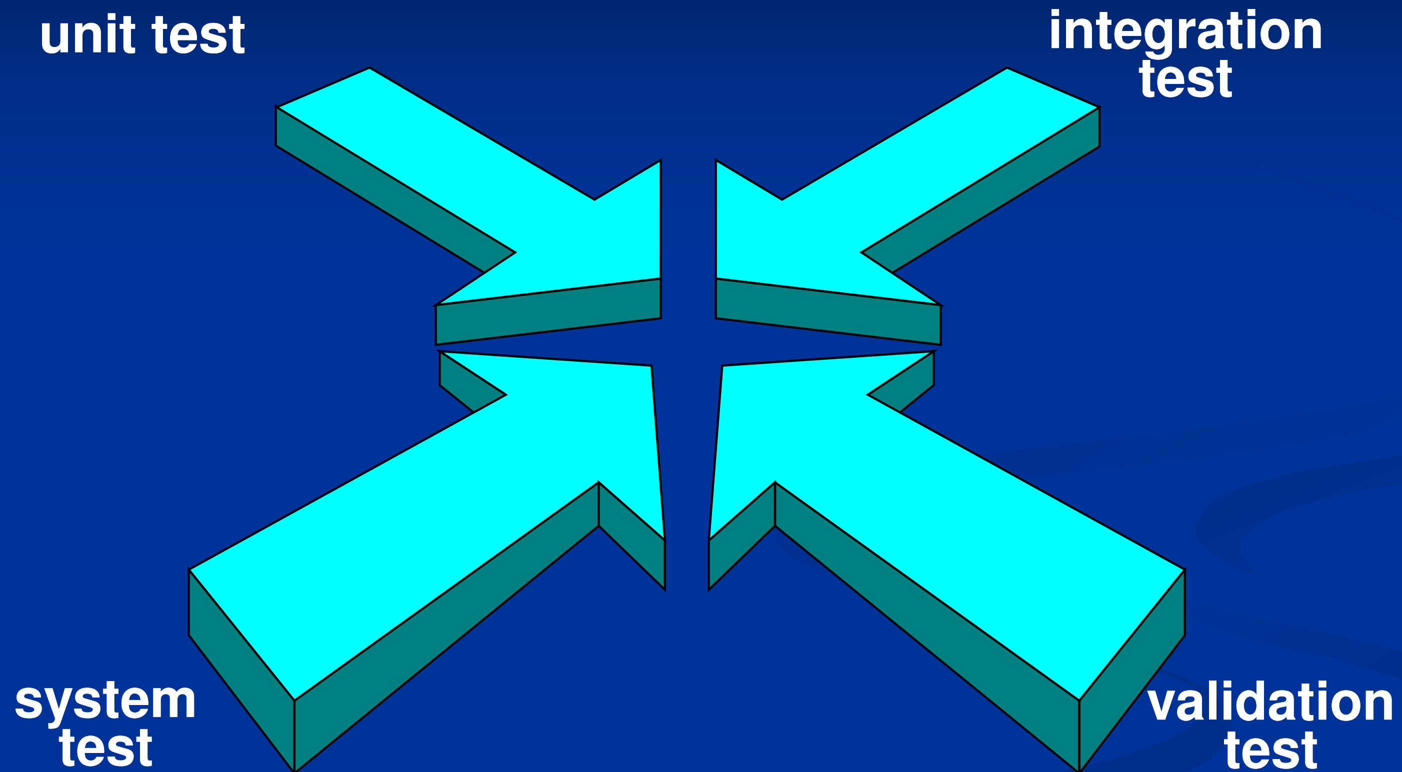
Understands the system
but, will test "gently"
and, is driven by "delivery"

independent tester

Must learn about the system,
but, will attempt to break it
and, is driven by quality

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Testing Strategy



These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Testing Strategy

- We begin by 'testing-in-the-small' and move toward 'testing-in-the-large'
- For conventional software:
  - The module (component) is our initial focus
  - Integration of modules follows
- For OO software:
  - our focus when "testing in the small" changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

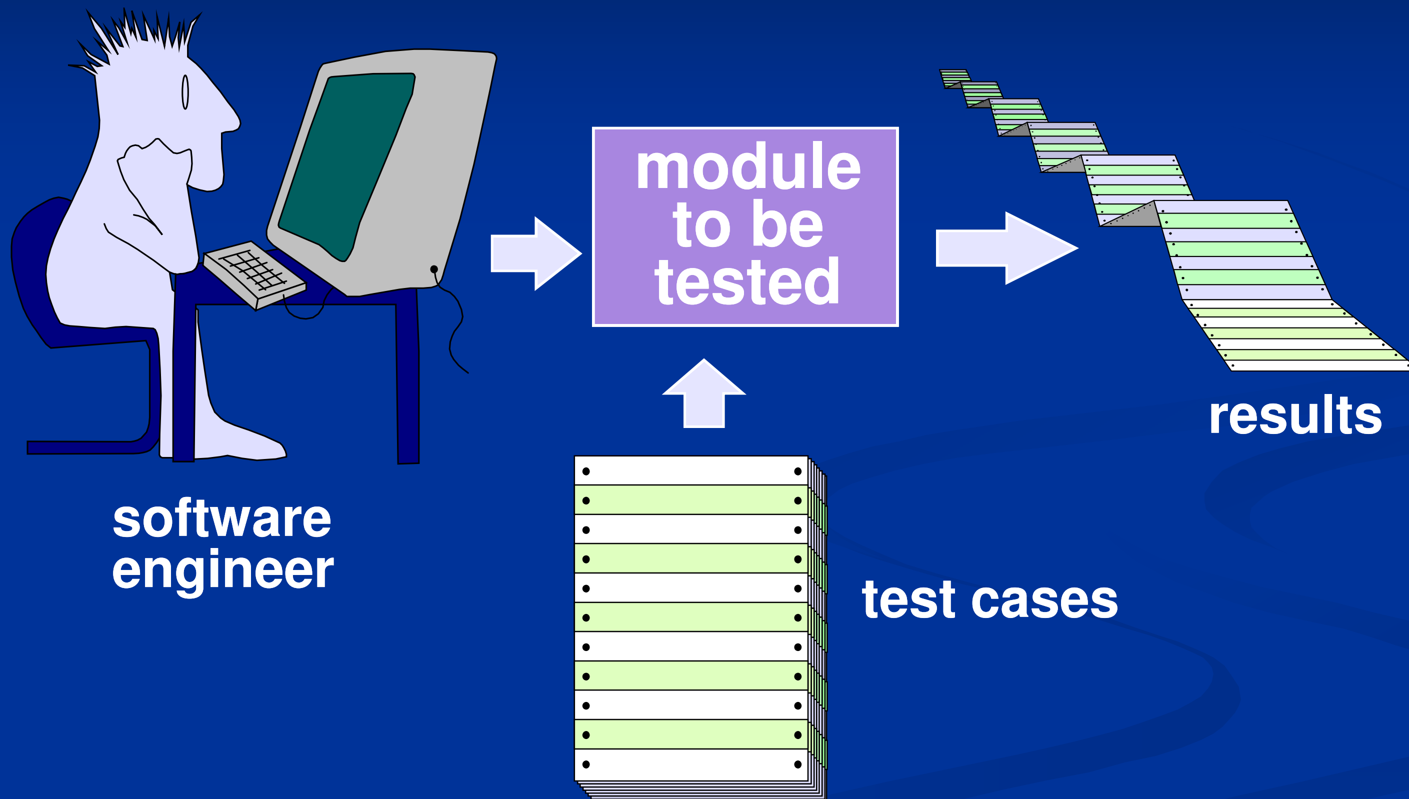
These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Strategic Issues

- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes "rapid cycle testing."
- Build "robust" software that is designed to test itself.
- Use effective formal technical reviews as a filter prior to testing.
- Conduct formal technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

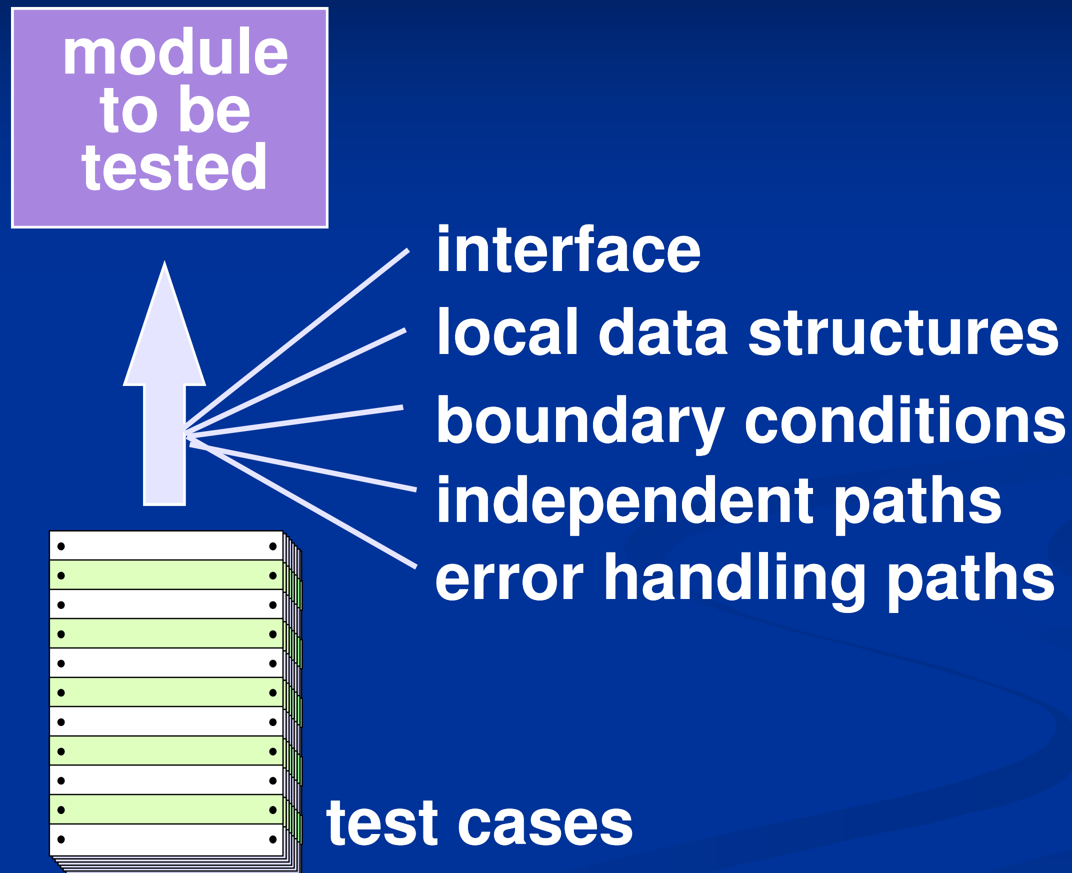
These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Unit Testing



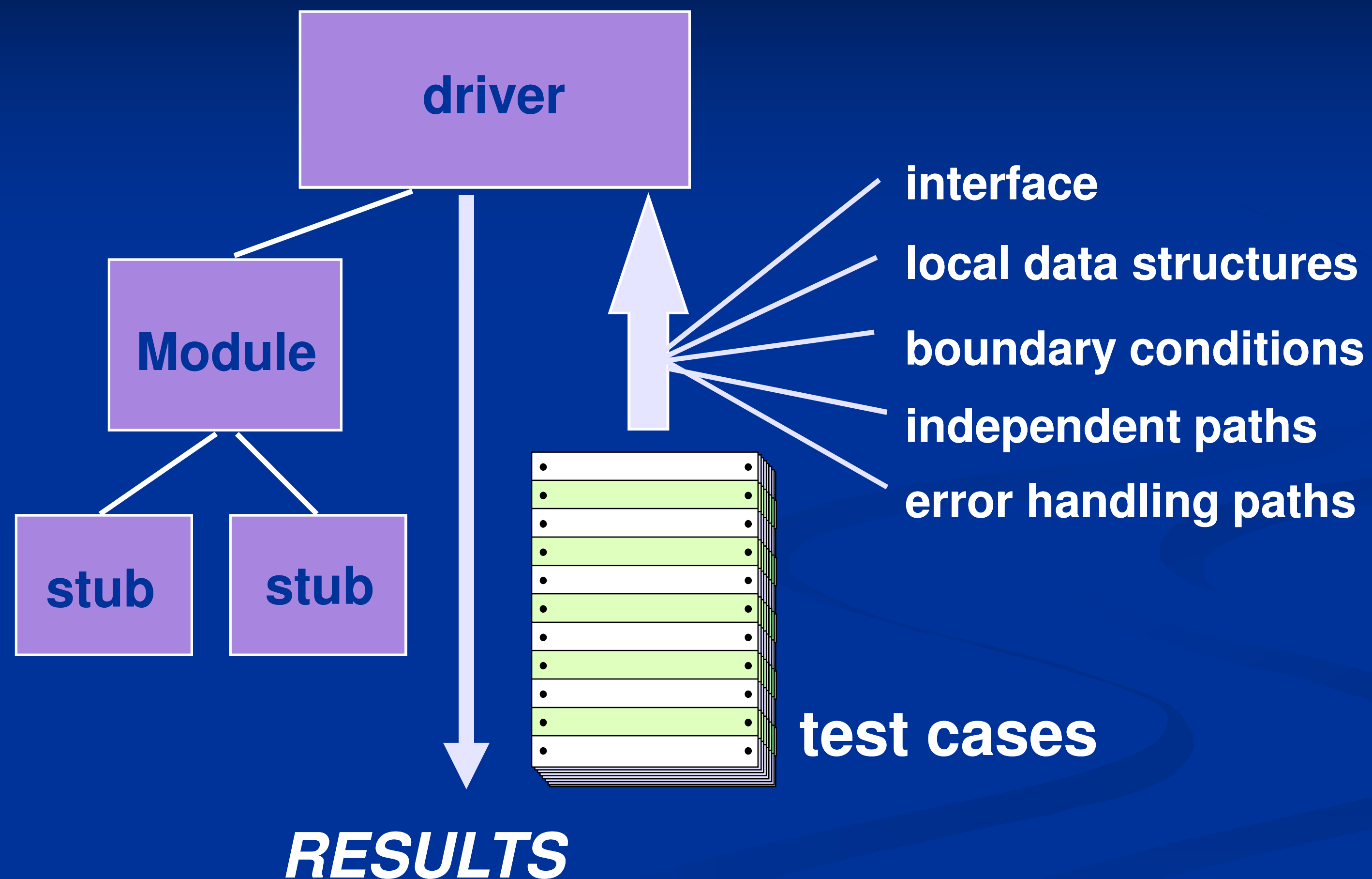
These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Unit Testing



These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Unit Test Environment

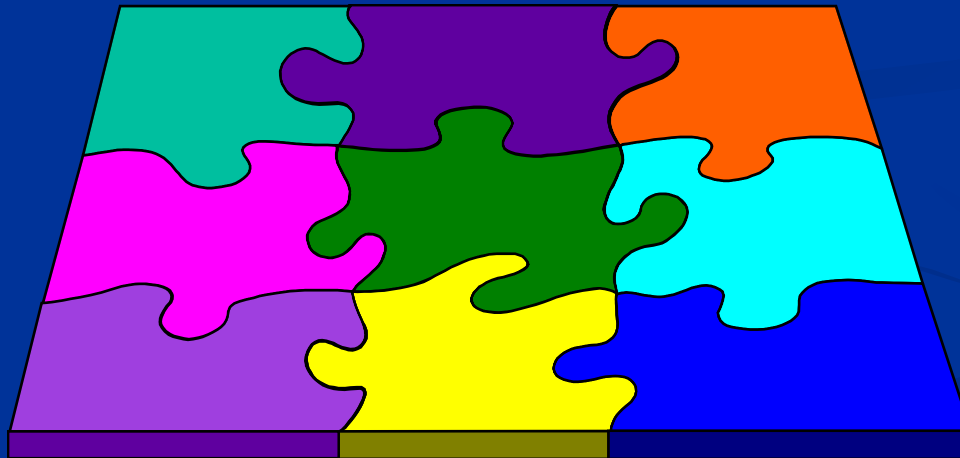


These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Integration Testing Strategies

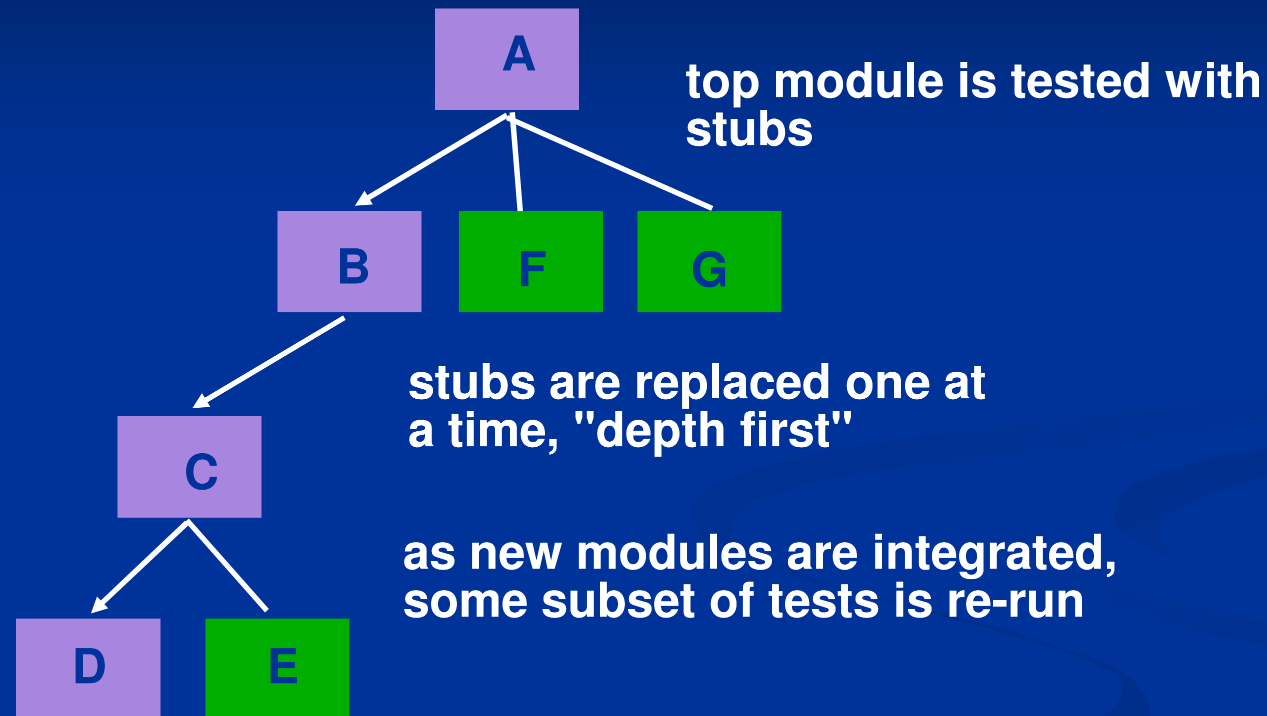
Options:

- **the “big bang” approach**
- **an incremental construction strategy**



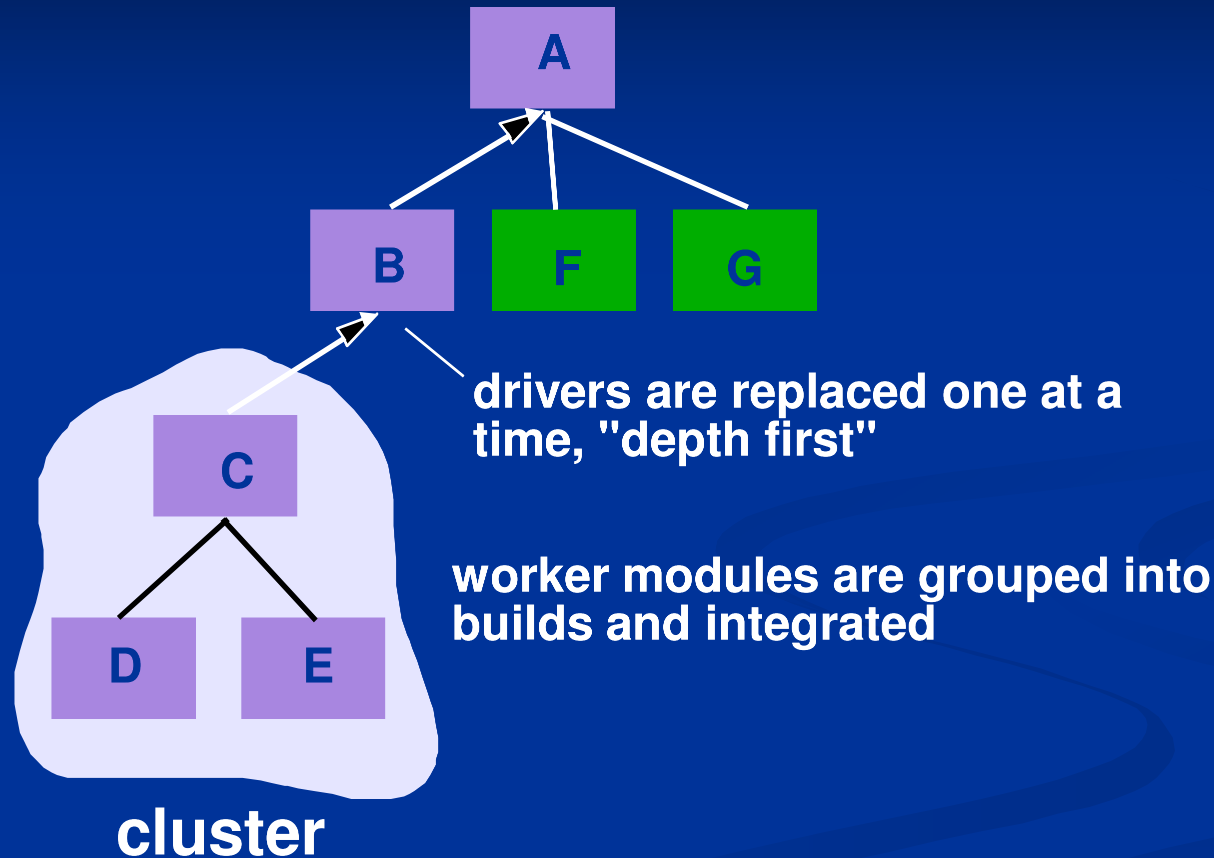
These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner’s Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Top Down Integration



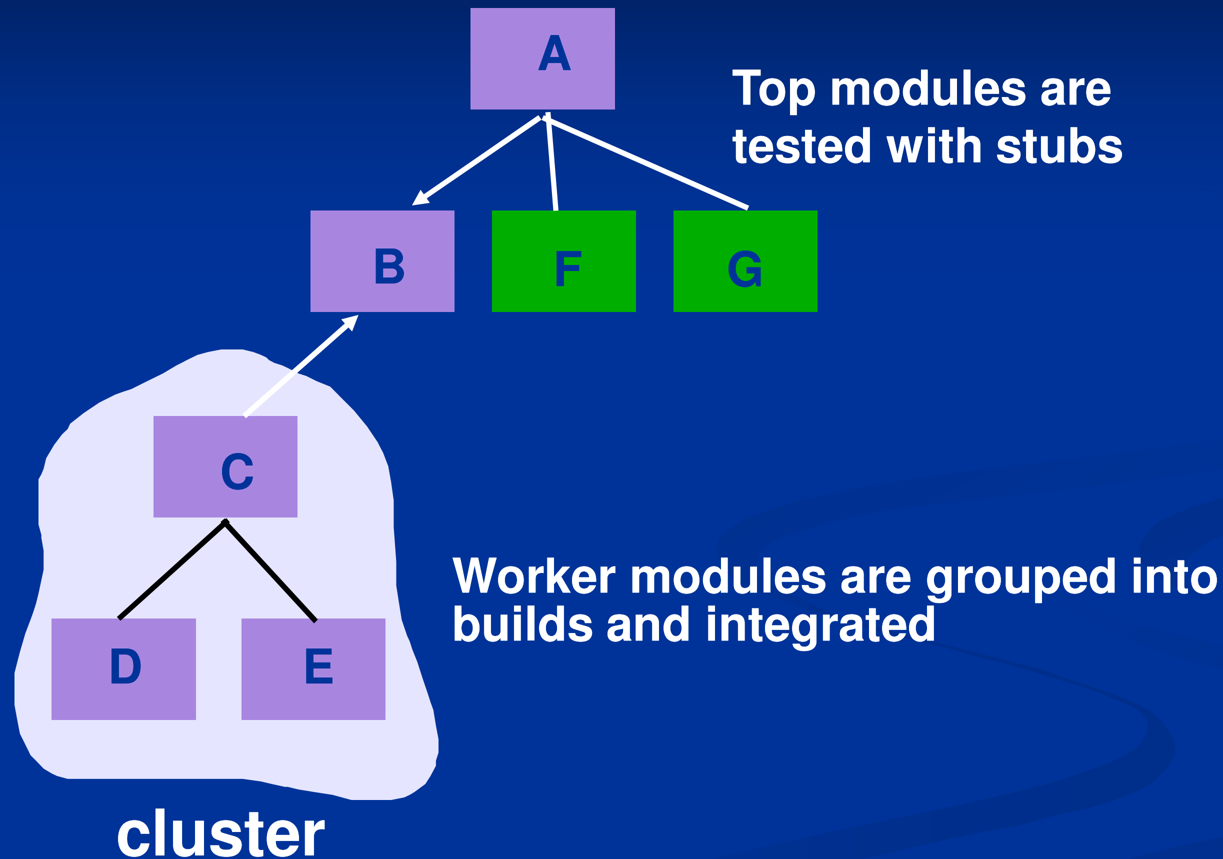
These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Bottom-Up Integration



These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Sandwich Testing



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Object-Oriented Testing

- Begins by evaluating the correctness and consistency of the OOA and OOD models
- Testing strategy changes
  - the concept of the ‘unit’ broadens due to encapsulation
  - integration focuses on classes and their execution across a ‘thread’ or in the context of a usage scenario
  - validation uses conventional black box methods
- Test case design draws on conventional methods, but also encompasses special features

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner’s Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Broadening the View of "Testing"

It can be argued that the review of OO analysis and design models is especially useful because the same semantic constructs (e.g., classes, attributes, operations, messages) appear at the analysis, design, and code level. Therefore, a problem in the definition of class attributes that is uncovered during analysis will circumvent side effects that might occur if the problem were not discovered until design or code (or even the next iteration of analysis).

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Testing the CRC Model

1. Revisit the CRC model and the object-relationship model.

2. Inspect the description of each CRC index card to determine if a delegated responsibility is part of the collaborator's definition.

3. Invert the connection to ensure that each collaborator that is asked for service is receiving requests from a reasonable source.

4. Using the inverted connections examined in step 3, determine whether other classes might be required or whether responsibilities are properly grouped among the classes.

5. Determine whether widely requested responsibilities might be combined into a single responsibility.

6. Steps 1 to 5 are applied iteratively to each class and through each evolution of the OOA model.

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

OOT Strategy

- Class testing is the equivalent of unit testing:
  - operations within the class are tested
  - the state behavior of the class is examined

- Integration applies three different strategies:
  - thread-based testing—integrates the set of classes required to respond to one input or event
  - use-based testing—integrates the set of classes required to respond to one use case
  - cluster testing—integrates the set of classes required to demonstrate one collaboration

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Smoke Testing

- A common approach for creating “daily builds” for product software
- Smoke testing steps:
  - Software components that have been translated into code are integrated into a “build.”
    - A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
  - A series of tests is designed to expose errors that will keep the build from properly performing its function.
    - The intent should be to uncover “show stopper” errors that have the highest likelihood of throwing the software project behind schedule.
  - The build is integrated with other builds and the entire product (in its current form) is smoke tested daily.
    - The integration approach may be top down or bottom up.

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner’s Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

High Order Testing

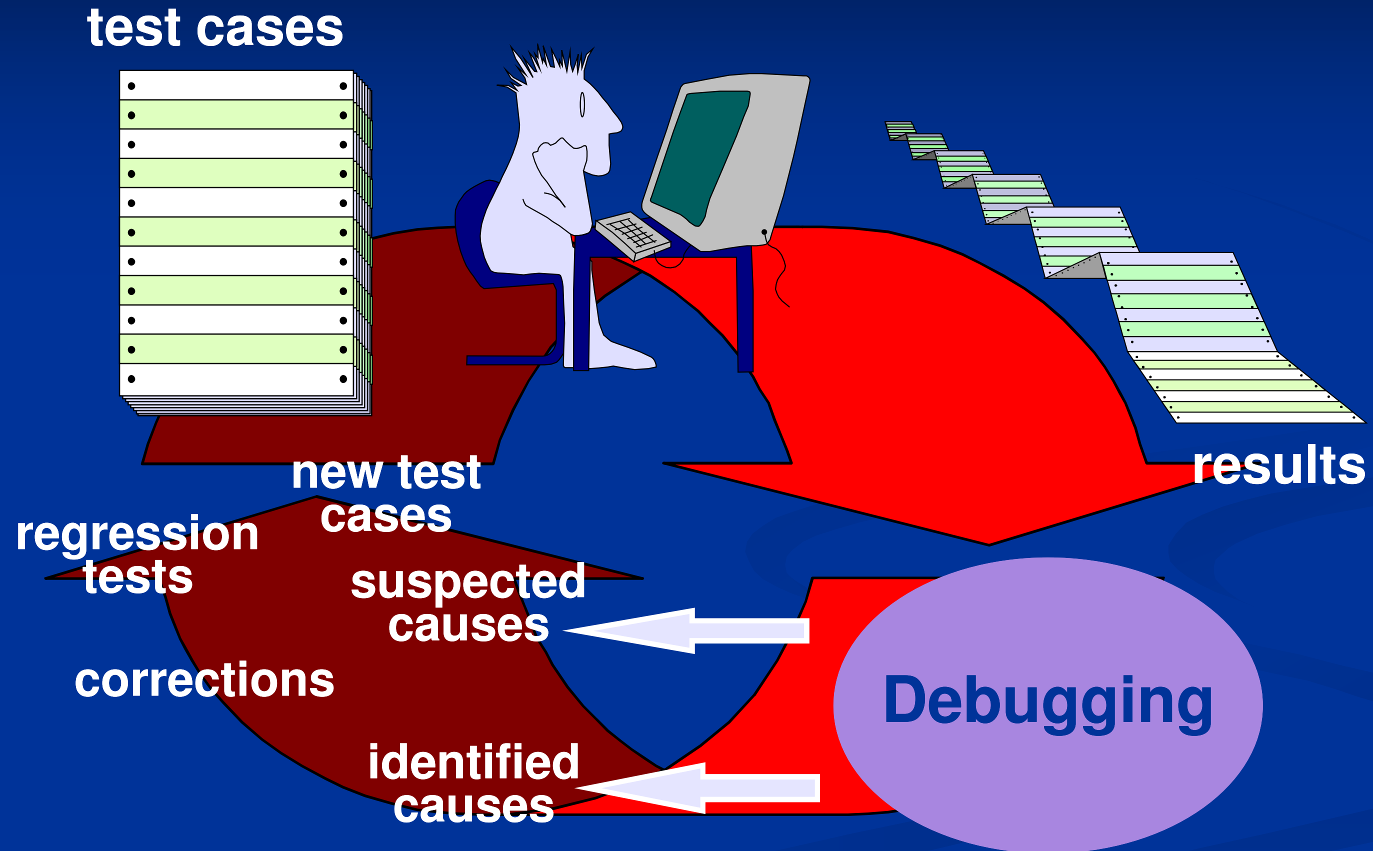
- Validation testing
  - Focus is on software requirements
- System testing
  - Focus is on system integration
- Alpha/Beta testing
  - Focus is on customer usage
- Recovery testing
  - forces the software to fail in a variety of ways and verifies that recovery is properly performed
- Security testing
  - verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration
- Stress testing
  - executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- Performance Testing
  - test the run-time performance of software within the context of an integrated system

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Debugging: A Diagnostic Process

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

The Debugging Process

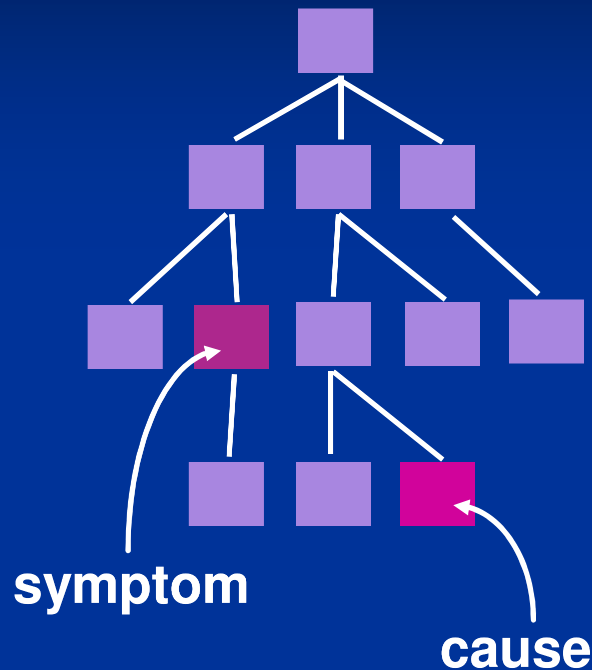


These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Debugging Effort

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

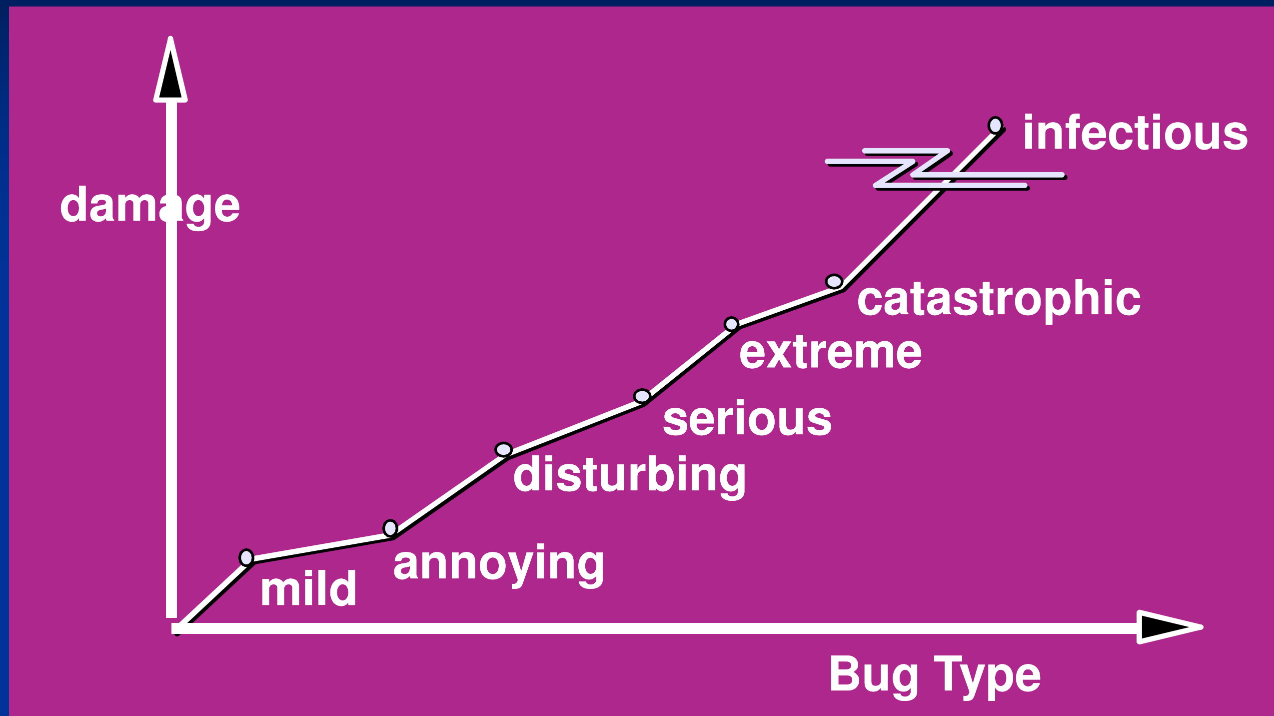
Symptoms & Causes



- Symptom and cause may be geographically separated
- Symptom may disappear when another problem is fixed
- Cause may be due to a combination of non-errors
- Cause may be due to a system or compiler error
- Cause may be due to assumptions that everyone believes
- Symptom may be intermittent

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Consequences of Bugs



Bug Categories: **function-related bugs, system-related bugs, data bugs, coding bugs, design bugs, documentation bugs, standards violations, etc.**

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Debugging Techniques

- Brute force / testing
- Backtracking
- Induction
- Deduction

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005

Debugging: Final Thoughts

1. Don't run off half-cocked, think about the symptom you're seeing.
2. Use tools (e.g., dynamic debugger) to gain more insight.
3. If at an impasse, get help from someone else.
4. Be absolutely sure to conduct regression tests when you do "fix" the bug.

These courseware materials are to be used in conjunction with *Software Engineering: A Practitioner's Approach,* 6/e and are provided with permission by R.S. Pressman & Associates, Inc., copyright © 1996, 2001, 2005