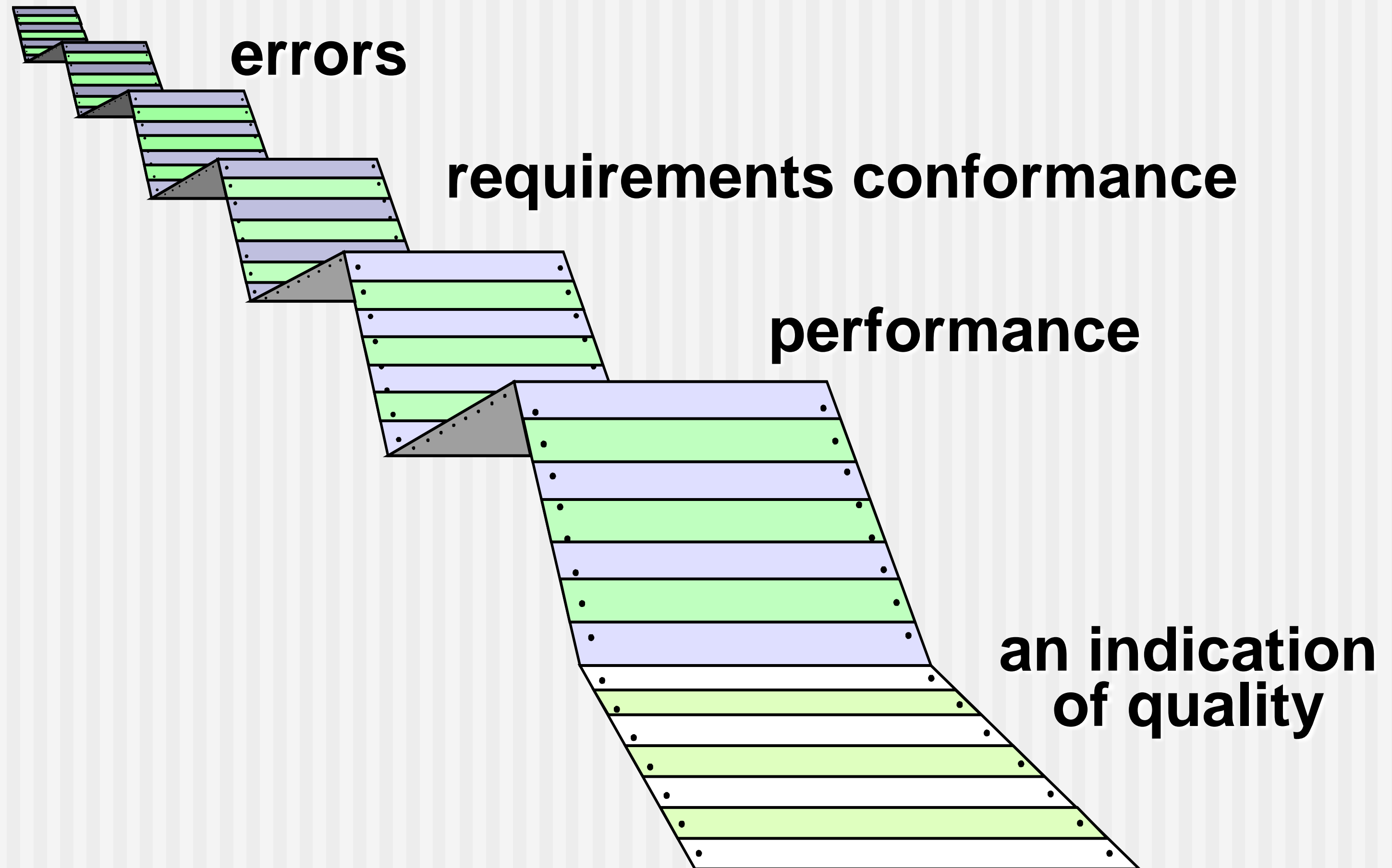
Software Testing

Dronacharya College of Engineering

Software Testing

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

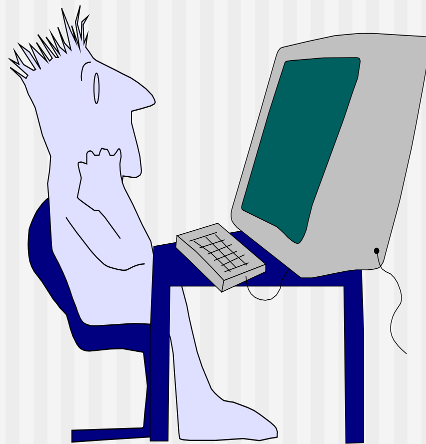
What Testing Shows



V & V

- *Verification* refers to the set of tasks that ensure that software correctly implements a specific function.
- *Validation* refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. Boehm [Boe81] states this another way:
  - *Verification:* "Are we building the product right?"
  - *Validation:* "Are we building the right product?"

Who Tests the Software?



developer

Understands the system

but, will test "gently"

and, is driven by "delivery"

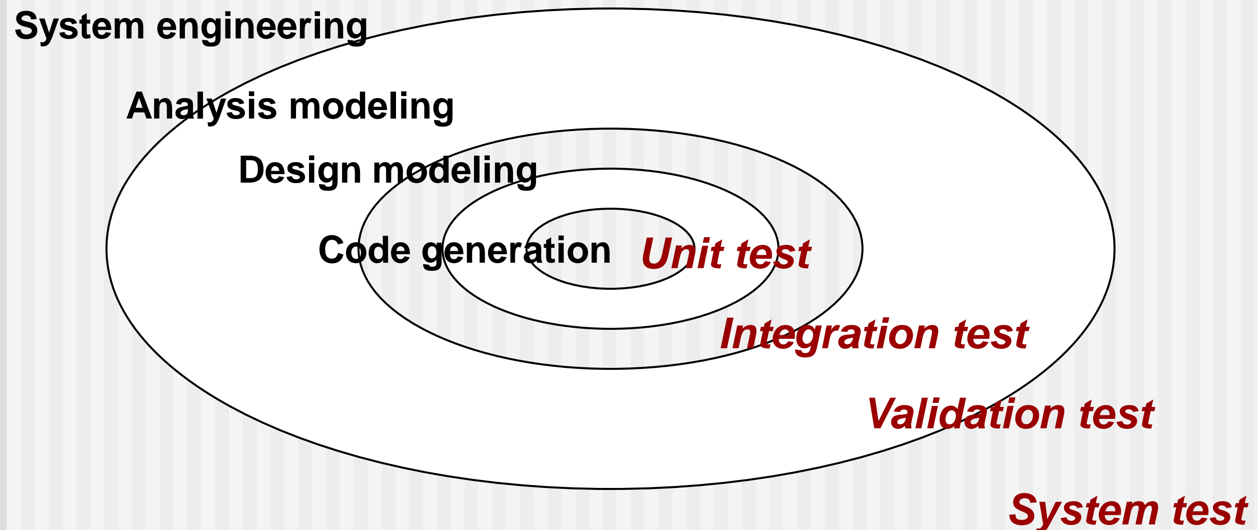
independent tester

Must learn about the system,

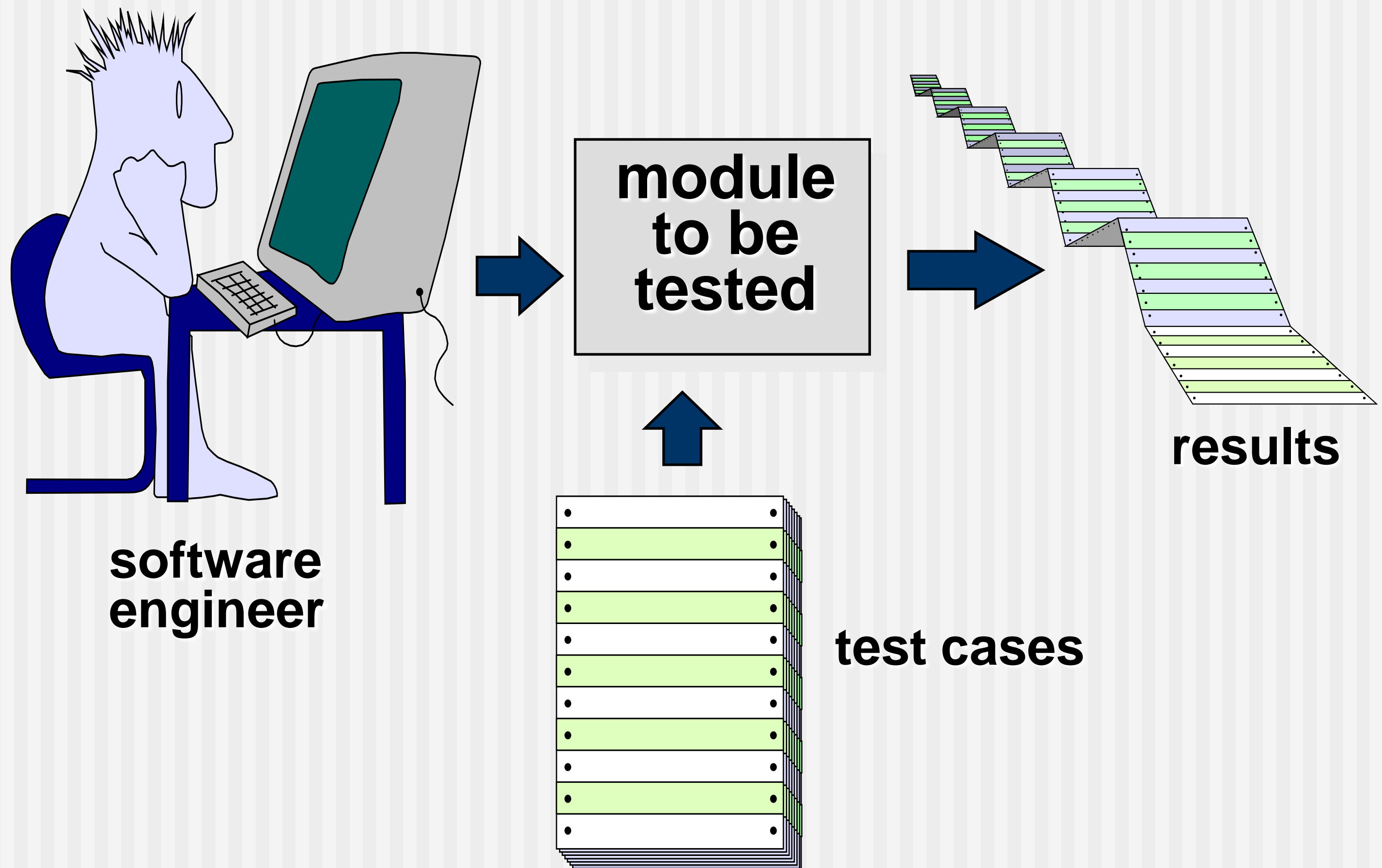
but, will attempt to break it

and, is driven by quality

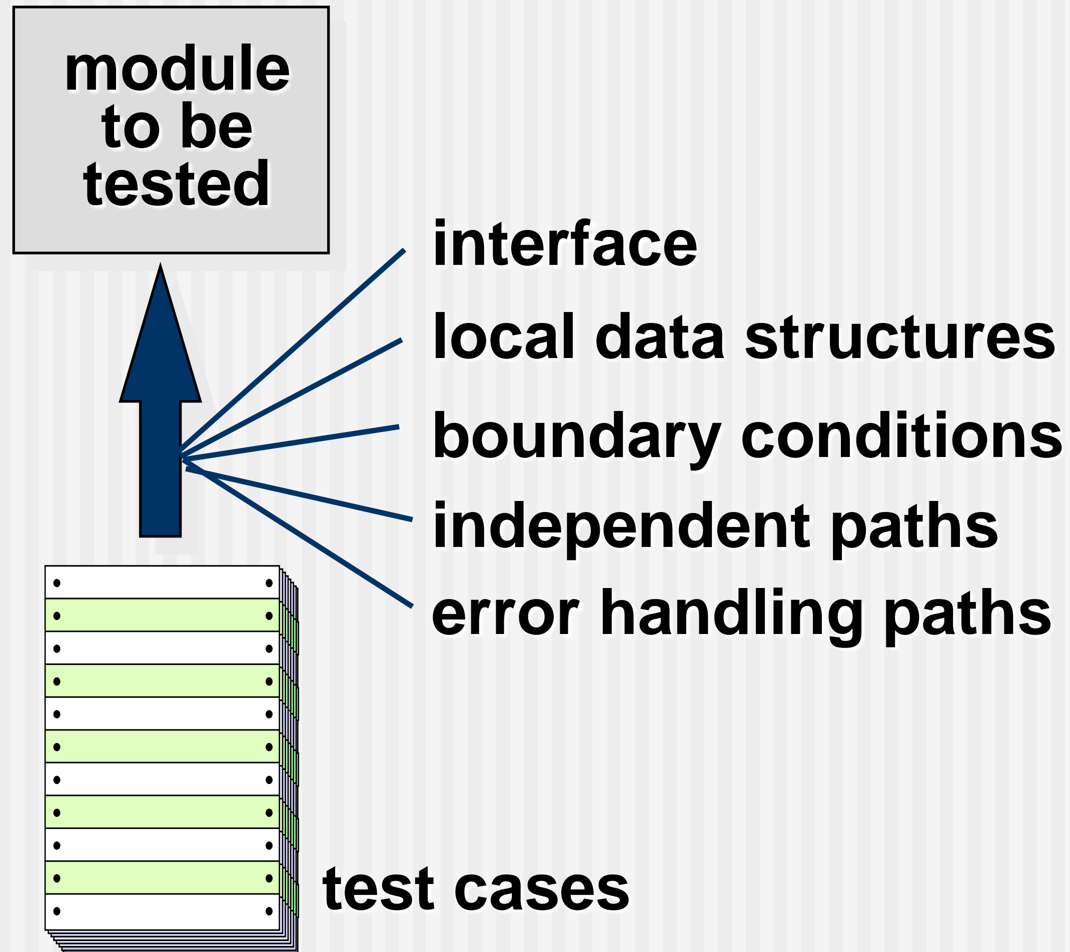
Testing Strategy



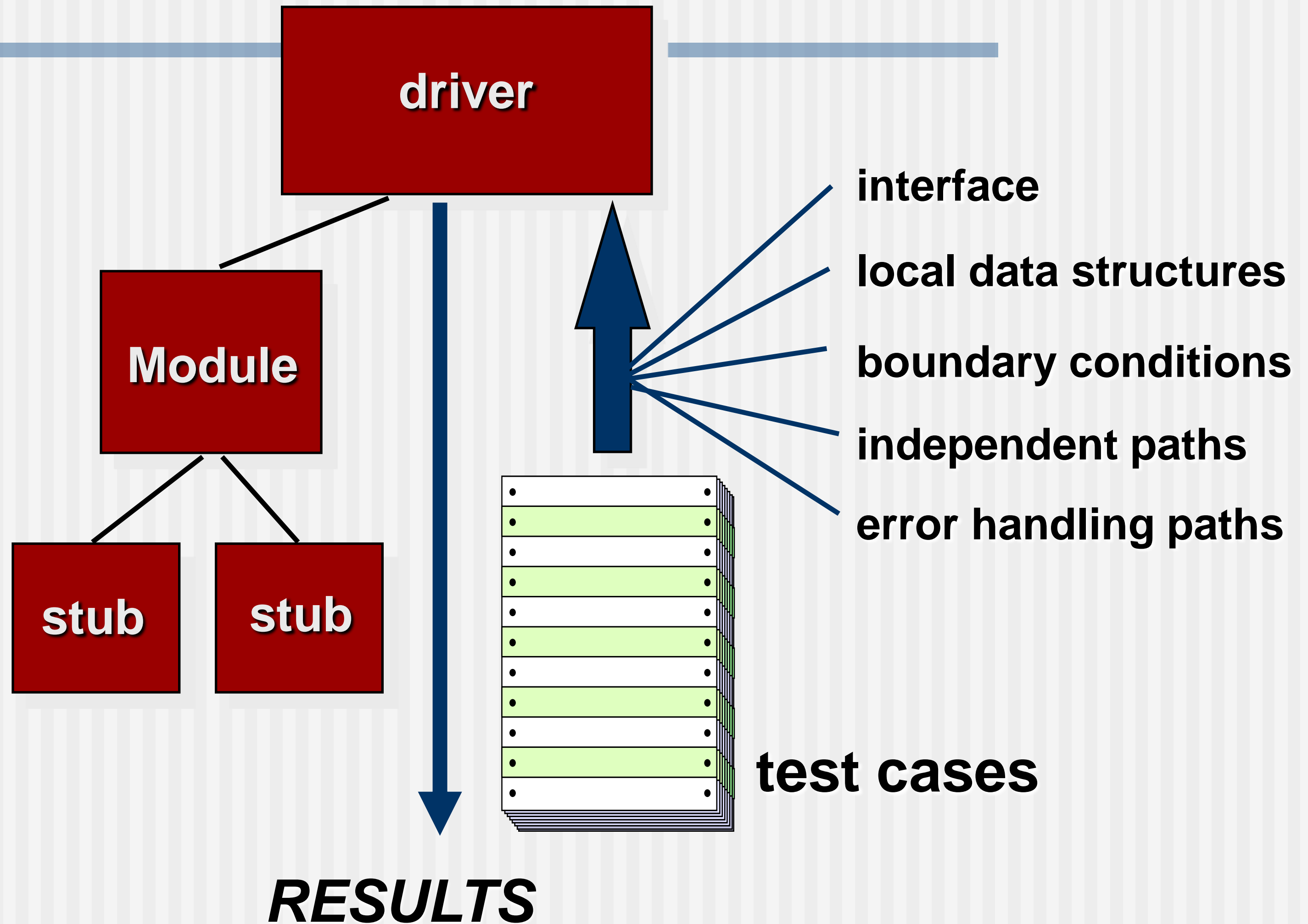
Unit Testing



Unit Testing



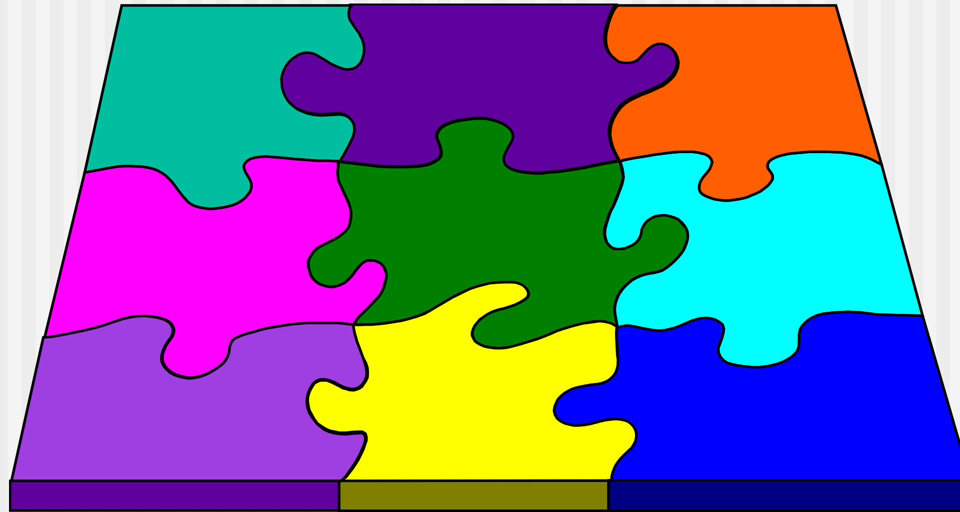
Unit Test Environment



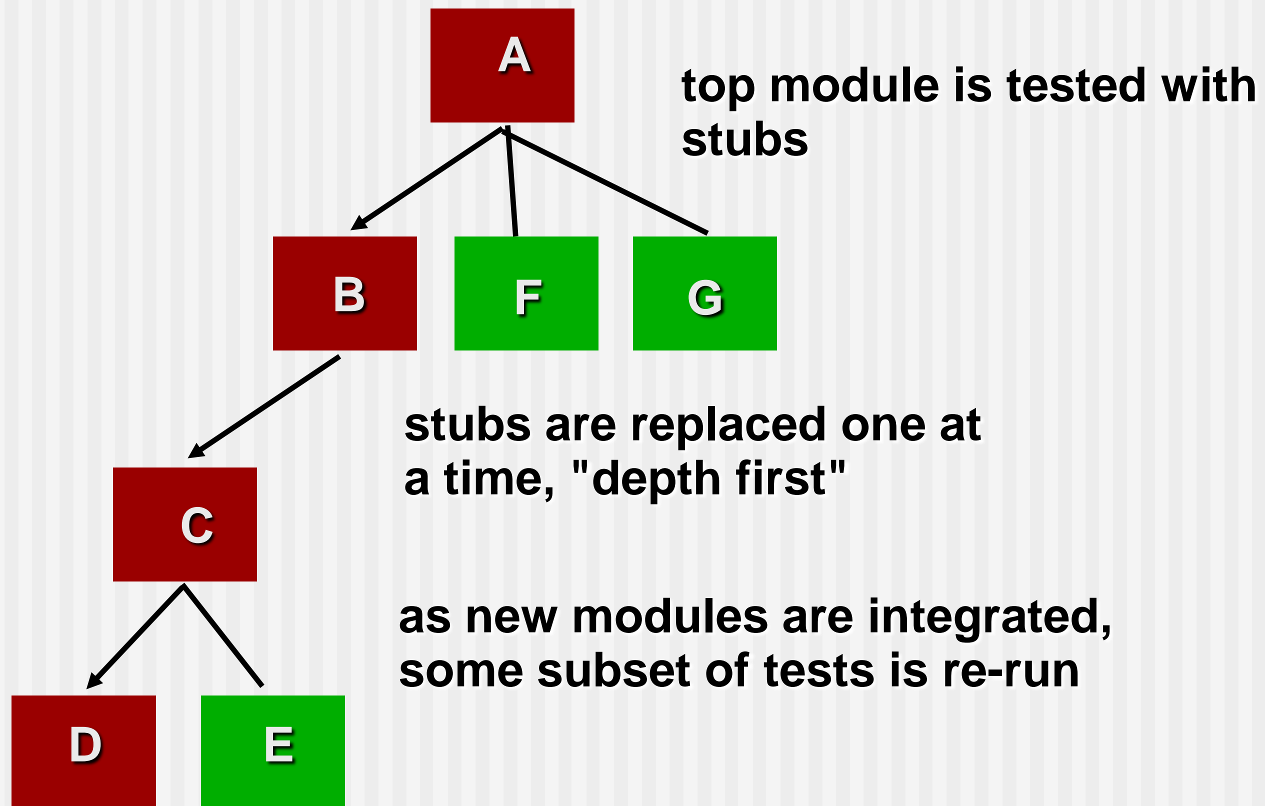
Integration Testing Strategies

Options:

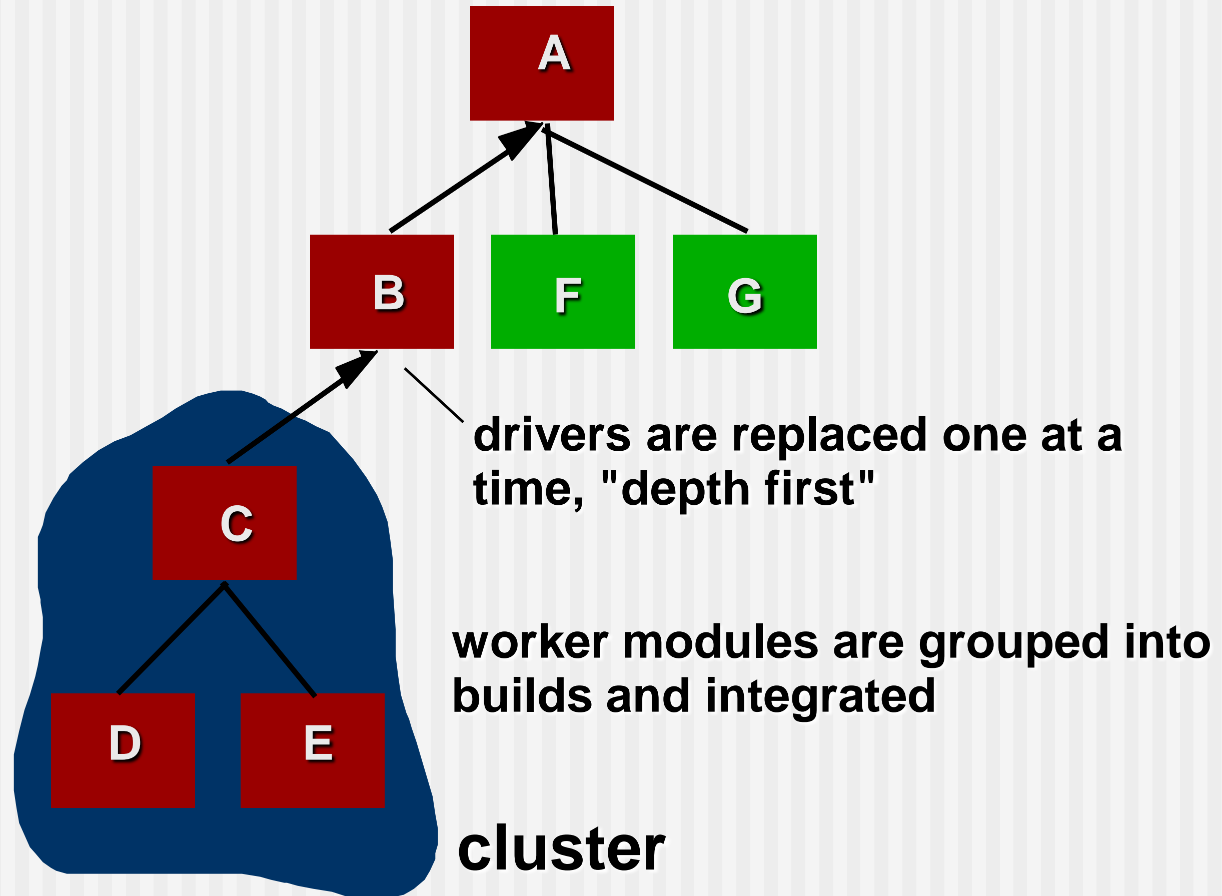
- **the “big bang” approach**
- **an incremental construction strategy**



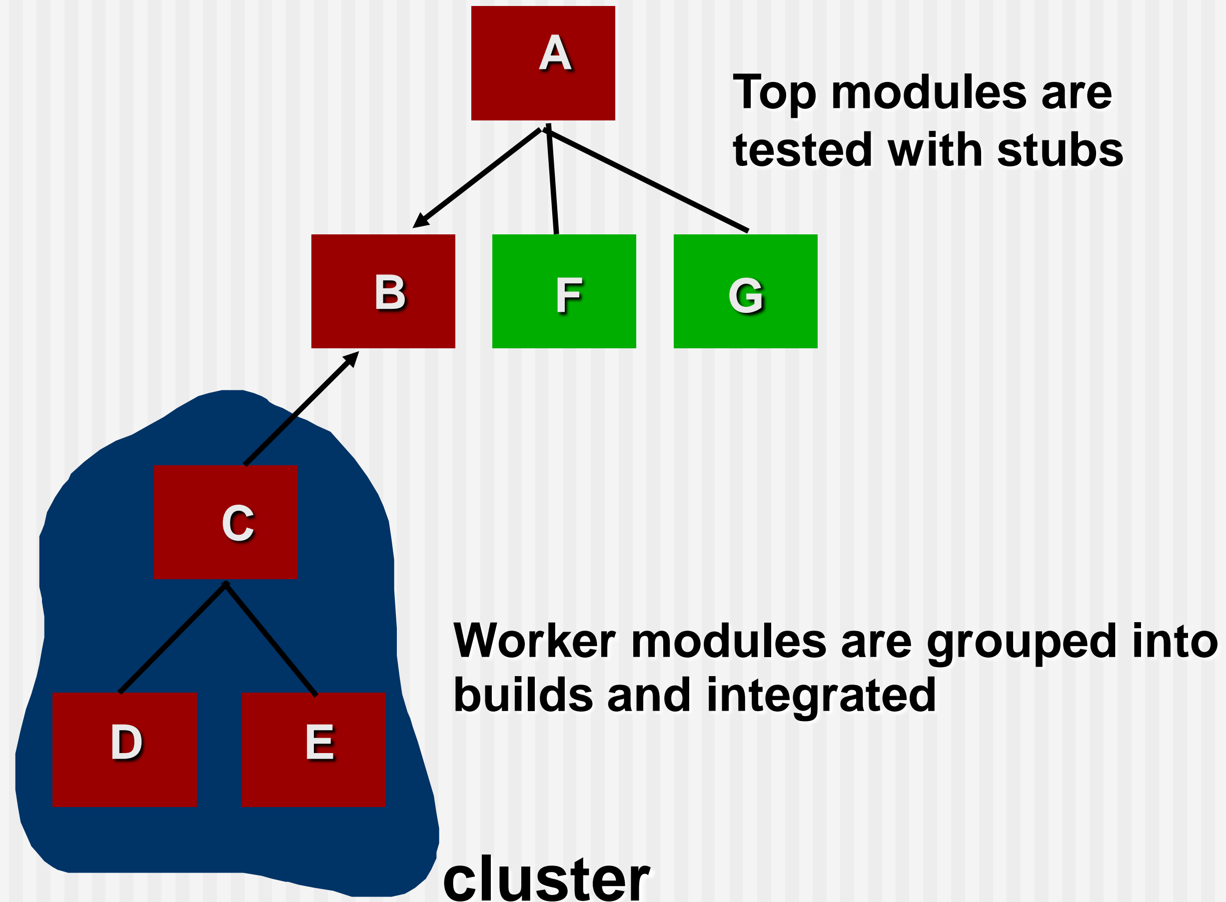
Top Down Integration



Bottom-Up Integration



Sandwich Testing



High Order Testing

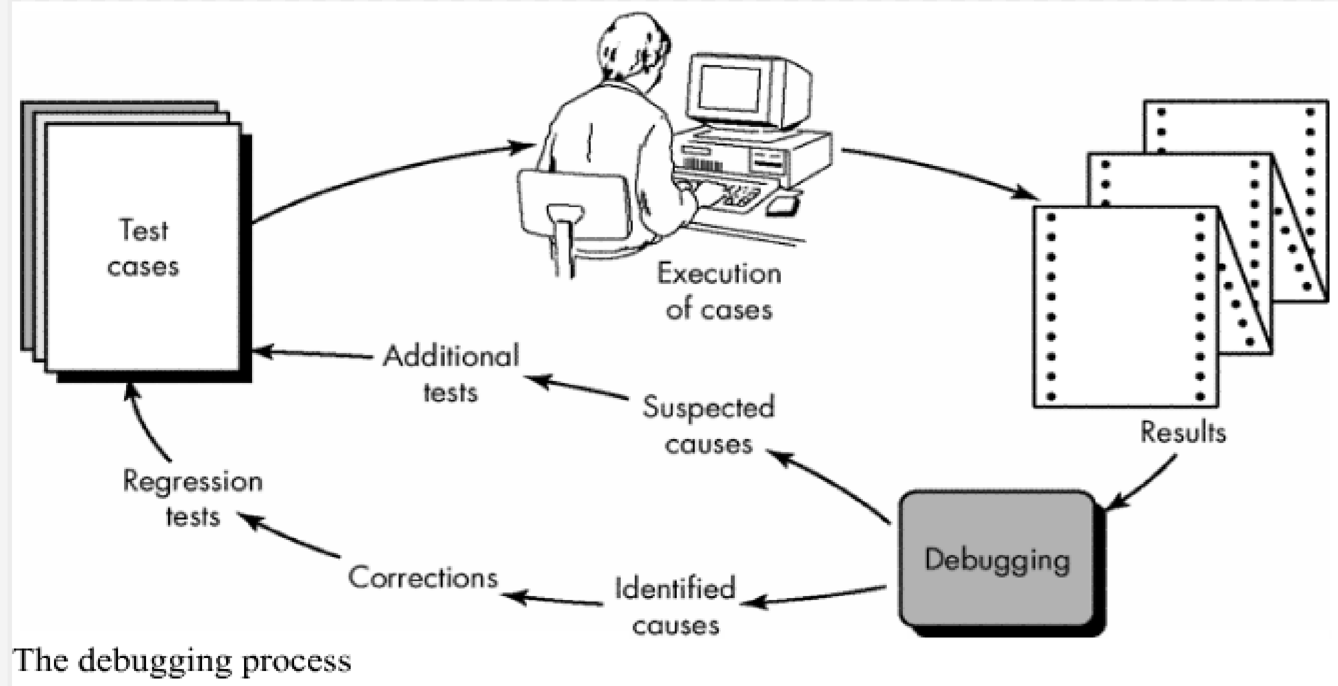
- Validation testing
  - Focus is on software requirements
- System testing
  - Focus is on system integration
- Alpha/Beta testing
  - Focus is on customer usage
- Recovery testing
  - forces the software to fail in a variety of ways and verifies that recovery is properly performed
- Security testing
  - 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

Debugging: A Diagnostic Process

The Art of Debugging

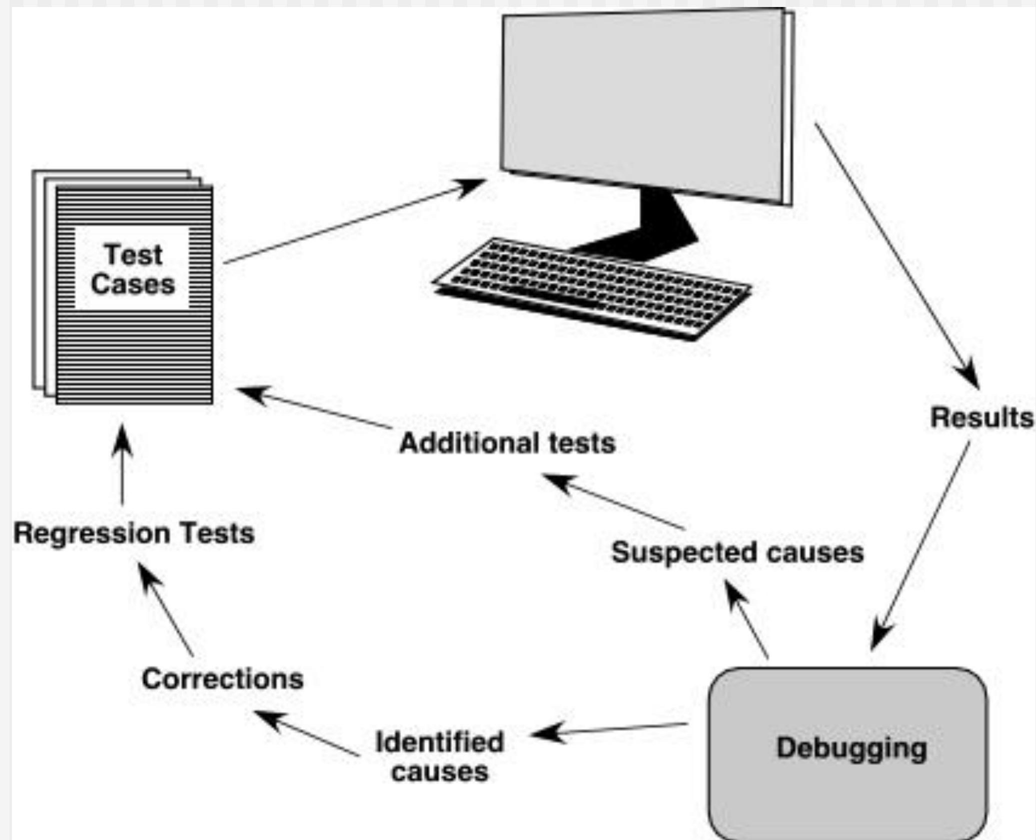
- Debugging occurs as a consequence of successful testing. That is, when a test case uncovers an error, debugging is the process that results in the removal of the error.

Debugging is not testing but always occurs as a consequence of testing.

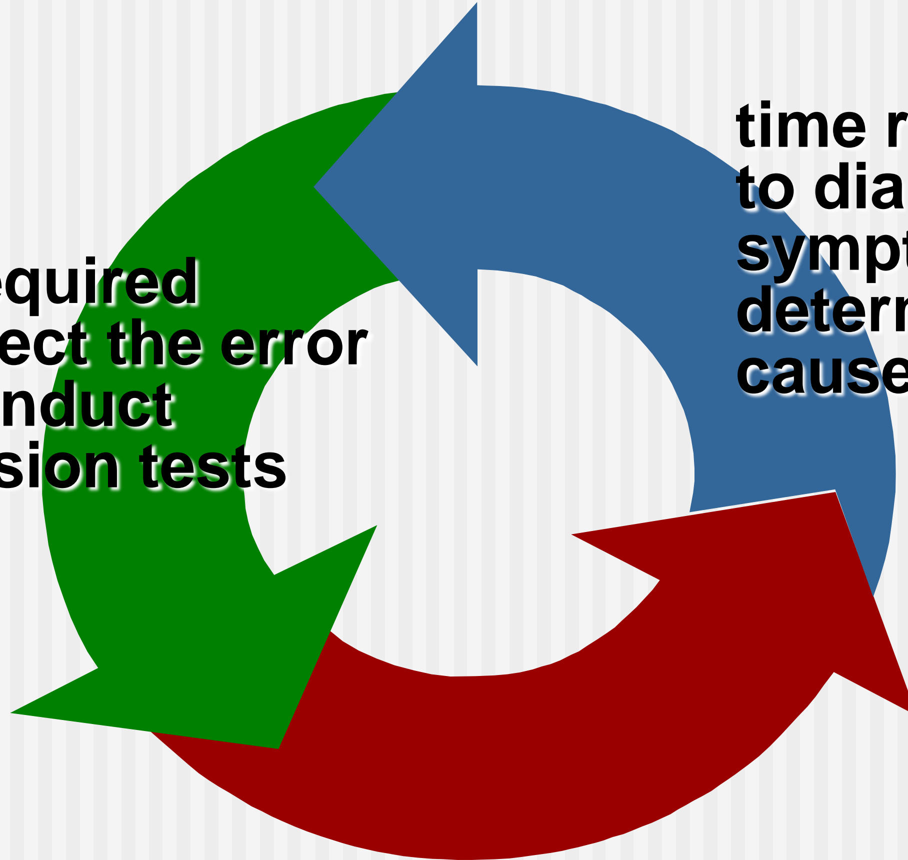


The debugging process

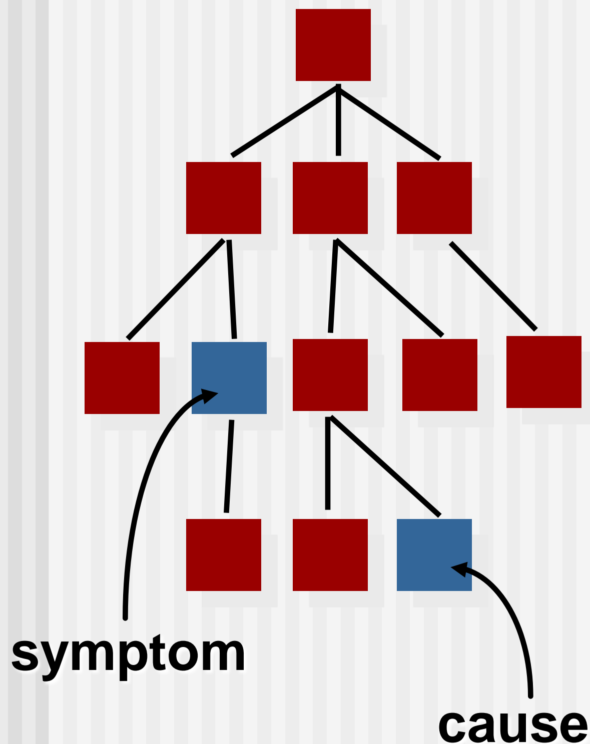
The Debugging Process



Debugging Effort

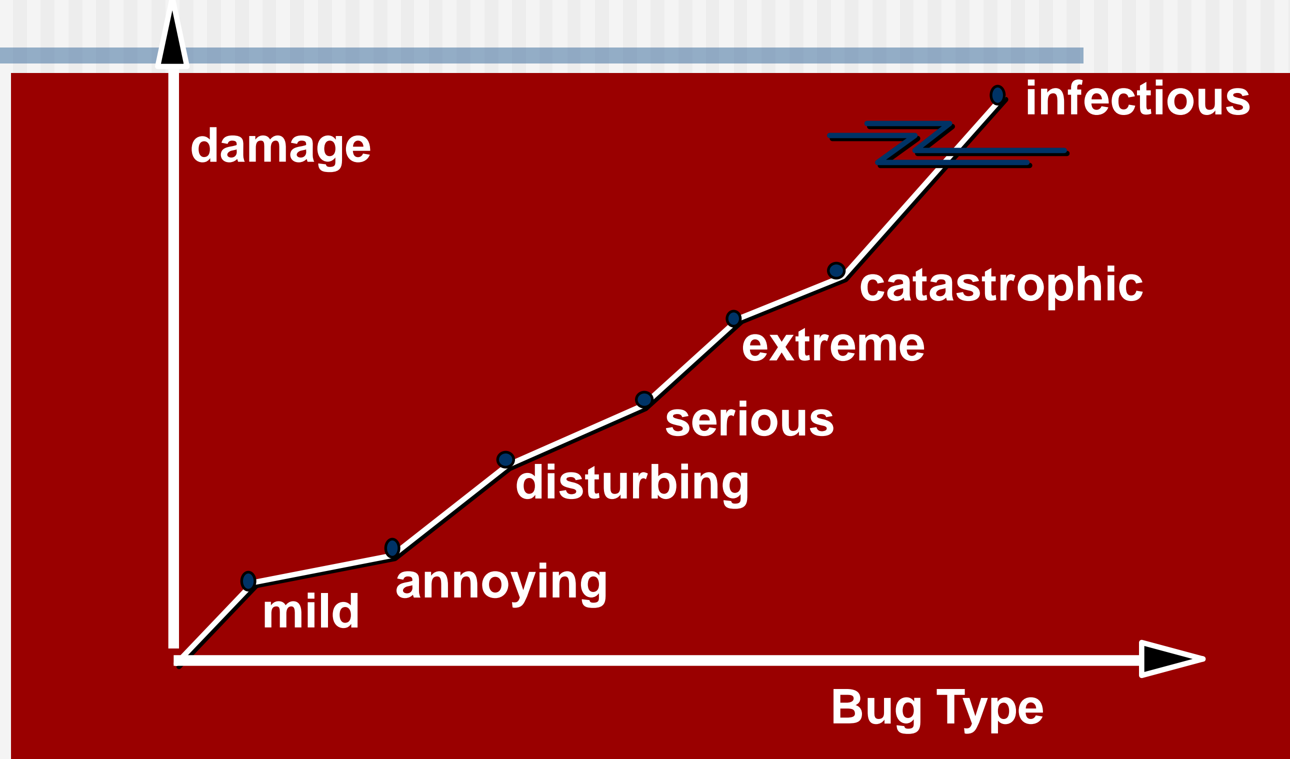


Symptoms & Causes



- 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

Consequences of Bugs



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

Debugging Techniques

- brute force / testing
- backtracking
- induction
- deduction

The Art of Debugging

- Three categories for debugging approaches may be proposed: (1) brute force, (2) backtracking, and (3) cause elimination.
- The **brute force** category of debugging is probably the most common and least efficient method for isolating the cause of a software error. We apply brute force debugging methods when all else fails.
- Using a "let the computer find the error" philosophy, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements. We hope that somewhere in the morass of information that is produced we will find a clue that can lead us to the cause of an error.

The Art of Debugging

- **Backtracking** is a fairly common debugging approach that can be used in small programs. Beginning at the site where a symptom has been uncovered, the source code is traced backward (manually) until the site of the cause is found. Unfortunately, as the number of source lines increases, the number of potential backward paths may become unmanageably large.
- **Cause elimination**. Data related to the error occurrence are organized to isolate potential causes. A "cause hypothesis" is devised and the aforementioned data are used to prove or disprove the hypothesis. Alternatively, a list of all possible causes is developed and tests are conducted to eliminate each.

Correcting the Error

Simple questions that should be asked before making the "correction" that removes the cause of a bug:

- *Is the cause of the bug reproduced in another part of the program?* In many situations, a program defect is caused by an erroneous pattern of logic that may be reproduced elsewhere.
- *What "next bug" might be introduced by the fix I'm about to make?* Before the correction is made, the source code (or, better, the design) should be evaluated to assess coupling of logic and data structures.
- *What could we have done to prevent this bug in the first place?* This question is the first step toward establishing a statistical software quality assurance approach. If you correct the process as well as the product, the bug will be removed from the current program and may be eliminated from all future programs.

Final Thoughts

- *Think* -- before you act to correct
- Use tools to gain additional insight
- If you're at an impasse, get help from someone else
- Once you correct the bug, use regression testing to uncover any side effects