Software Testing Strategies

Ch. Md. Rakin Haider

Software Testing

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

Software Testing Approach

- To perform effective testing, you should conduct effective technical reviews. By doing this, many errors will be eliminated before testing commences.
- Testing begins at the component level and works “outward” toward the integration of the entire computer-based system.
- Different testing techniques are appropriate for different software engineering approaches and at different points in time.
- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- Testing and debugging are different activities,

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

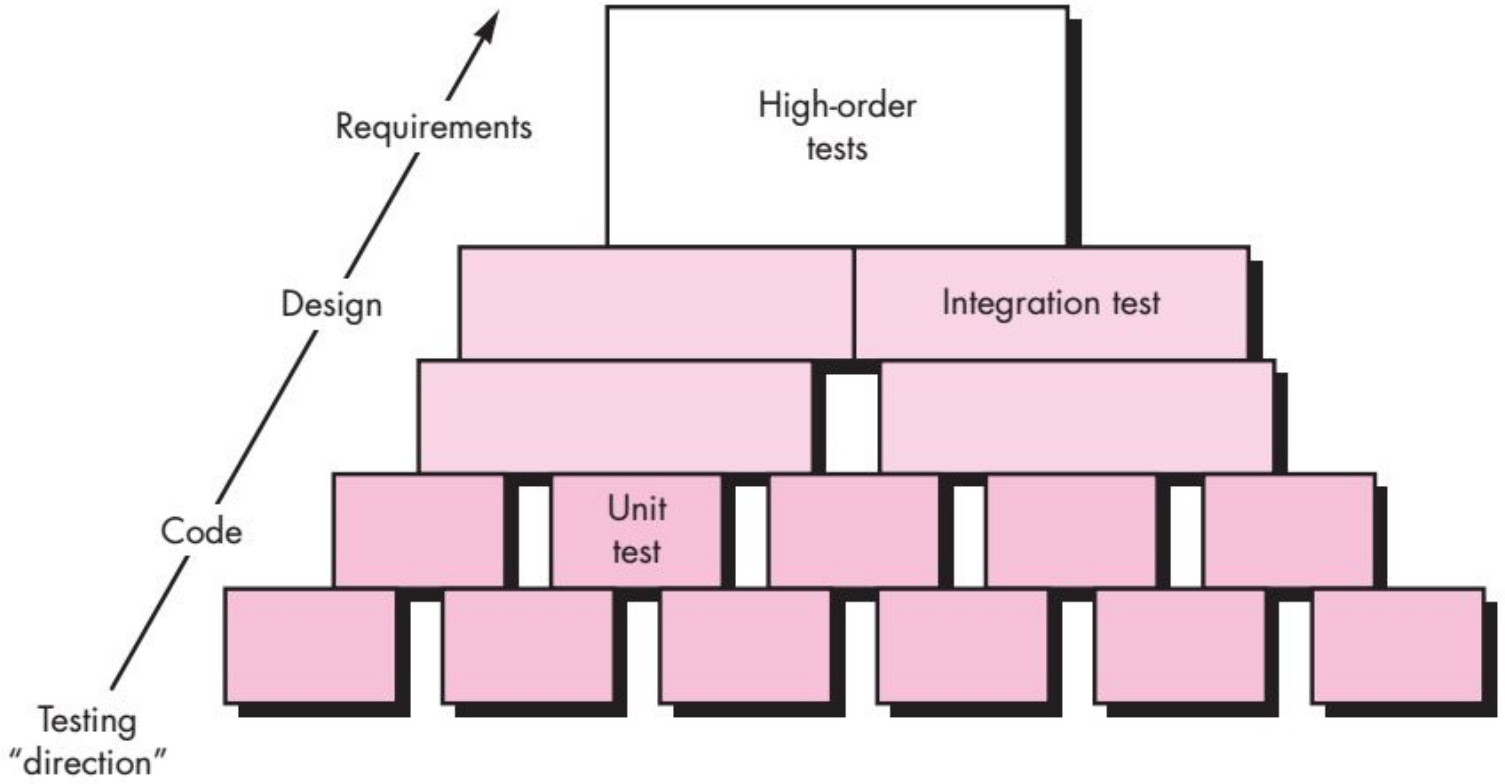
but, will attempt to break it

and, is driven by quality

Testing Strategy

- 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

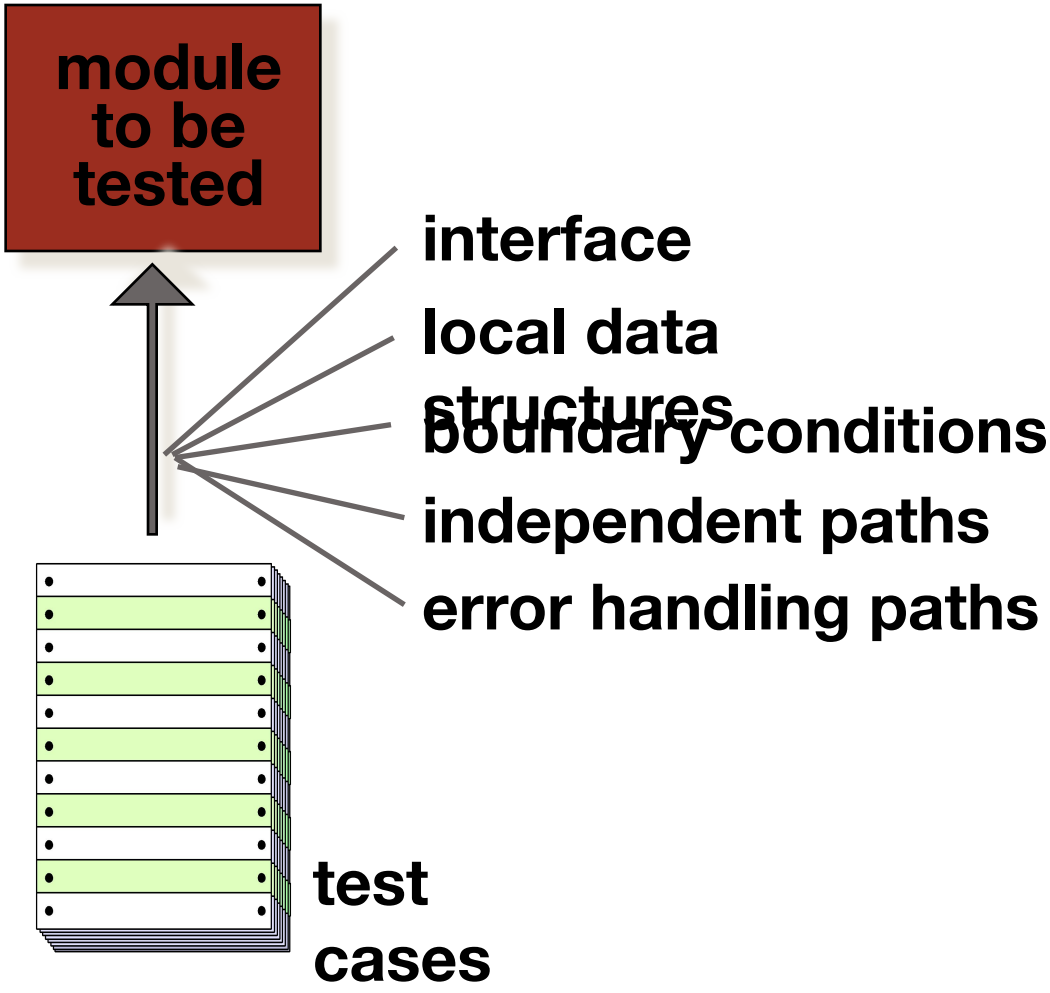
Software Testing Steps



Unit Testing

- *Unit testing* focuses verification effort on the smallest unit of software design—the software component or module.

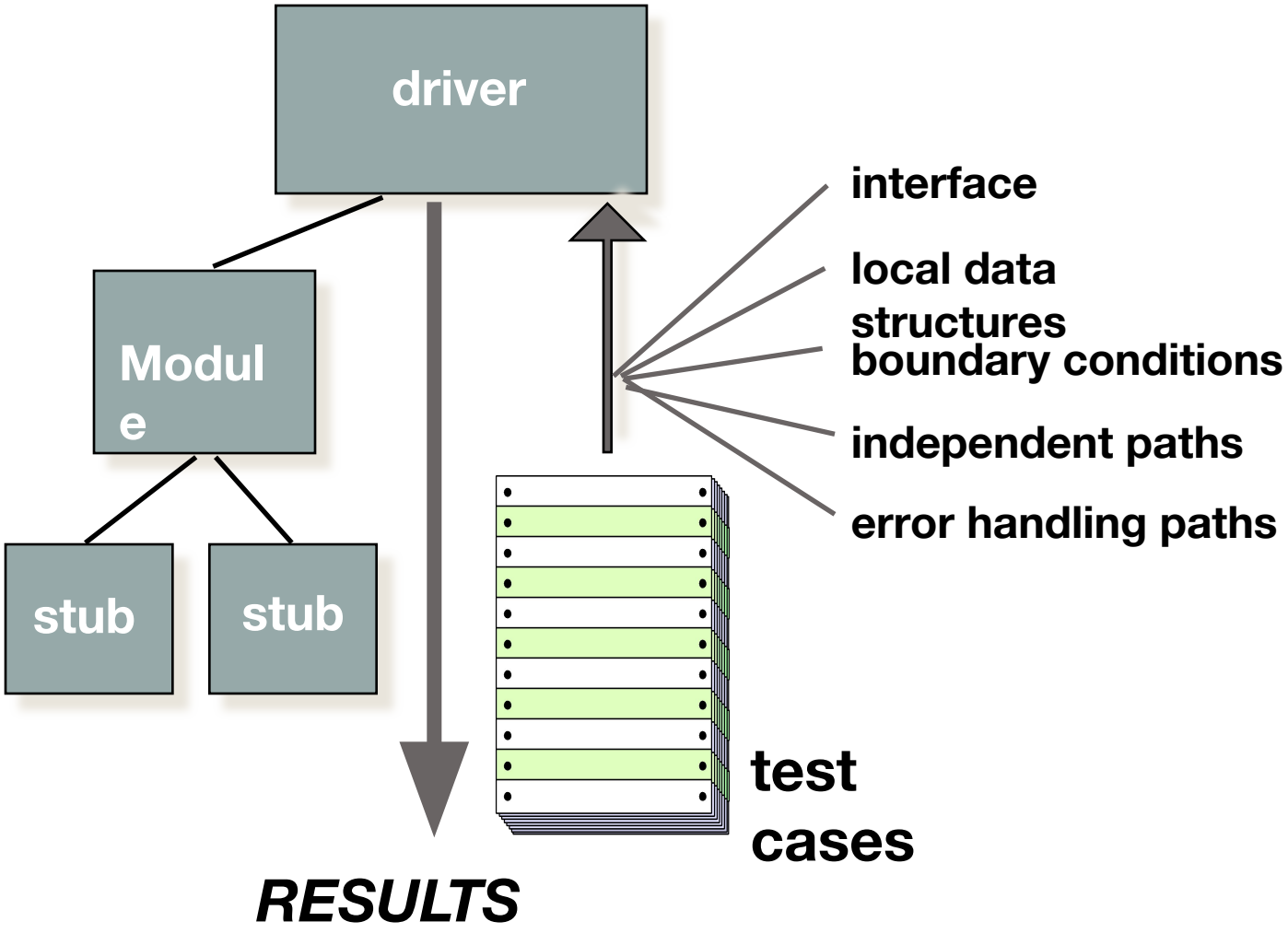
Unit-test considerations



Unit-test procedures

- In most applications a *driver* is nothing more than a "main program" that accepts test case data, passes such data to the component (to be tested), and prints relevant results.
- *Stubs* serve to replace modules that are subordinate (invoked by) to the component to be tested.
- A stub or "dummy subprogram" uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.

Unit-test procedures



Integration Testing

- If they all work individually, why do you doubt that
  they wont work when we put them together ?? In short “Why integration testing is required ??”
  - Data can be lost across an interface
  - One component can have an inadvertent, adverse effect on another
  - Sub-functions, when combined, may not produce the desired major function
  - Individually acceptable imprecision may be magnified to unacceptable levels
  - Global data structures can present problems.

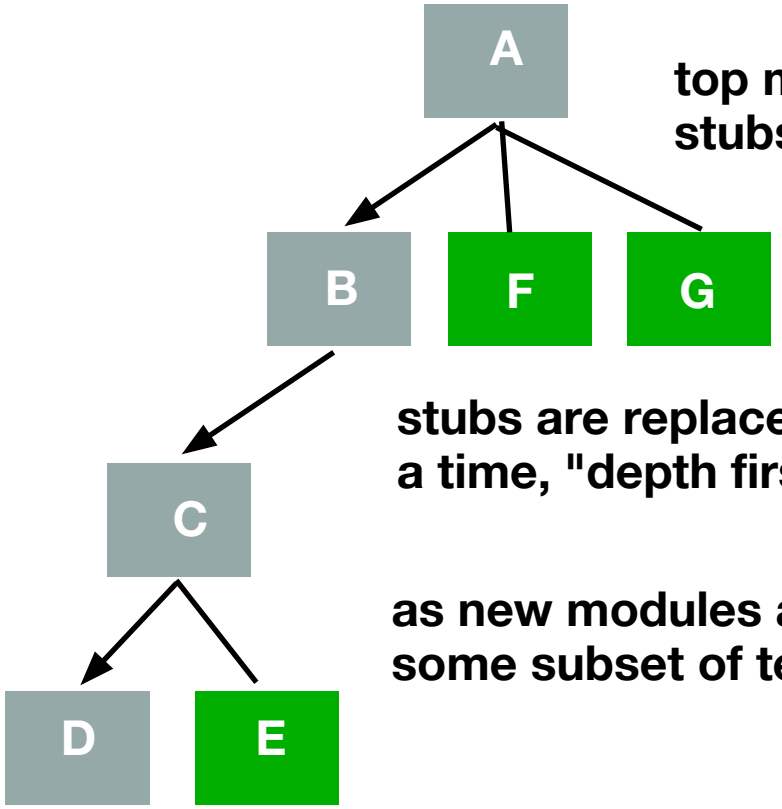
Integration testing

- Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.
- A major problem during system integration is localising errors.
- To simplify error localisation, systems should be incrementally integrated.

Integration Testing Strategies

- Options:
  - the “big bang” approach
    - All components are combined in advance.
  - An incremental construction strategy
    - Top-down integration.
    - Bottom-up integration.

Top Down Integration



top module is tested with stubs

stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

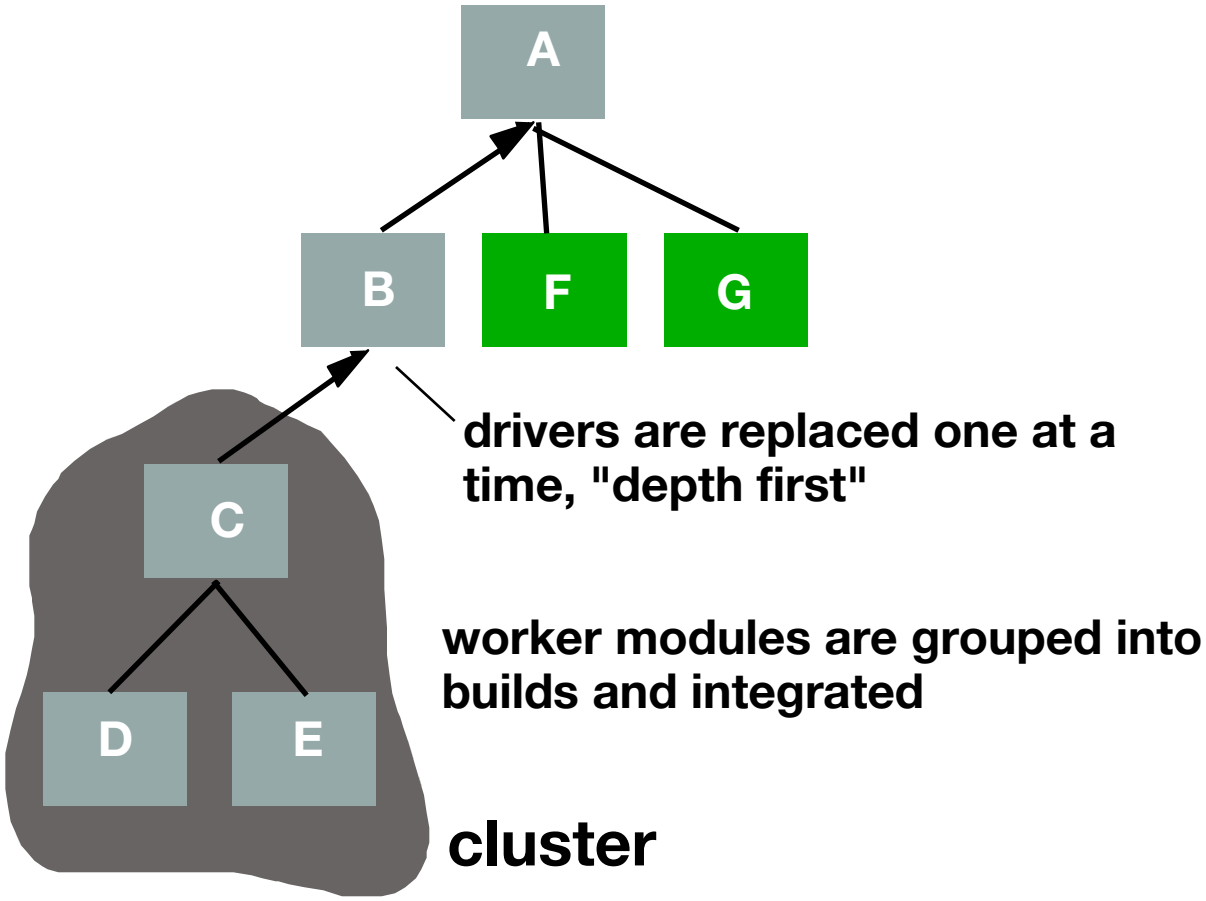
Top Down Integration

- Depth-First Integration
  - *Depth-first integration* integrates all components on a major control path of the program structure.
- Breadth-First Integration
  - *Breadth-first integration* incorporates all components directly subordinate at each level, moving across the structure horizontally.

Top Down Integration Steps

- The main control module is used as a test driver, and stubs are substituted for all components directly subordinate to the main control module.
- Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
- Tests are conducted as each component is integrated.
- On completion of each set of tests, another stub is replaced with the real component.
- Regression testing (discussed later) may be

Bottom-Up Integration



Bottom Up Testing Steps

- Low-level components are combined into clusters (sometimes called *builds*) that perform a specific software sub-function.
- A *driver* (a control program for testing) is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.

Regression Testing

- *Regression testing* is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects
- Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed.
- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors.
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.

Smoke Testing

- *Smoke testing* is an integration testing approach that is commonly used when product software is developed

- A common approach for creating "daily builds" for product software

Smoke Testing Steps

- 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.

VALIDATION TESTING

- *Validation testing* begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package, and interfacing errors have been uncovered and corrected.
  - Alpha and Beta Testing

Alpha and Beta Testing

- The *alpha test* is conducted at the developer's site by a representative group of end users.

- The *beta test* is conducted at one or more end-user sites. Unlike alpha testing, the developer generally is not present.
  - A variation on beta testing, called *customer acceptance testing*, is sometimes performed when custom software is delivered to a customer under contract.

Other Types of Testing

- System Testing
  - Recovery Testing
  - Stress Testing
  - Security Testing
  - Deployment Testing
  - Performance Testing

- P.S. No need to go into details.

Debugging

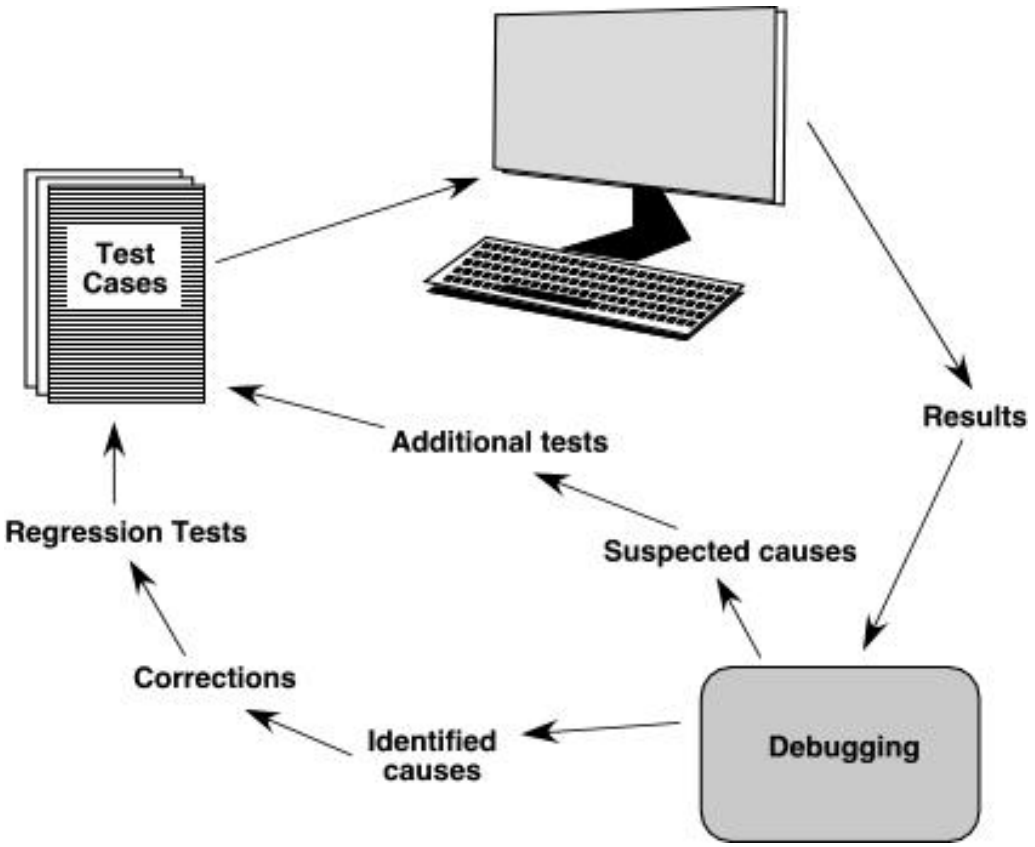
- Are there any difference between debugging and testing? Or are they two different names of the same thing ?

Debugging

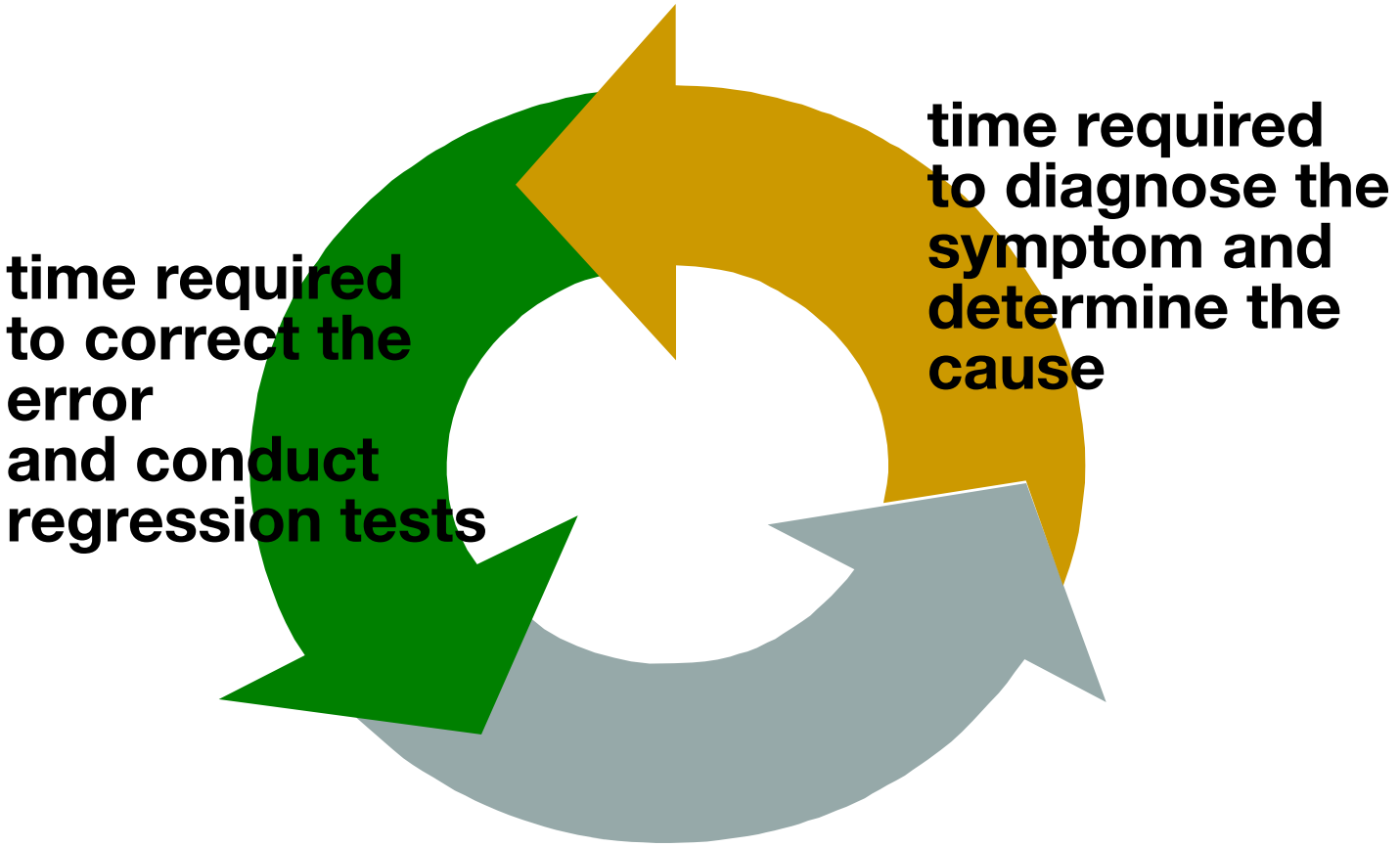
- *Debugging* occurs as a consequence of successful testing.

- That is, when a test case uncovers an error, debugging is the process that results in the removal of the error.

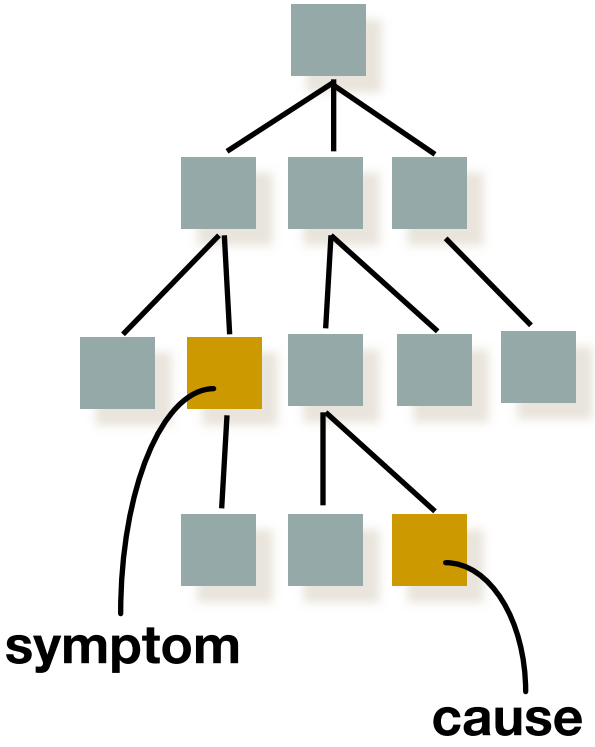
The Debugging Process



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

Debugging Techniques

- Brute force
- Backtracking
- Cause Elimination

Reference

- *Software Engineering: A Practitioner's Approach, 7/e*
- **by Roger S. Pressman**
- **Chapter 17**

Thank You