Software Testing

"Testing is the process of executing a program with the intention of finding errors." – Myers

"Testing can show the presence of bugs but never their absence." - Dijkstra

Strategic 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, but debugging must be accommodated in any testing strategy. am

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,

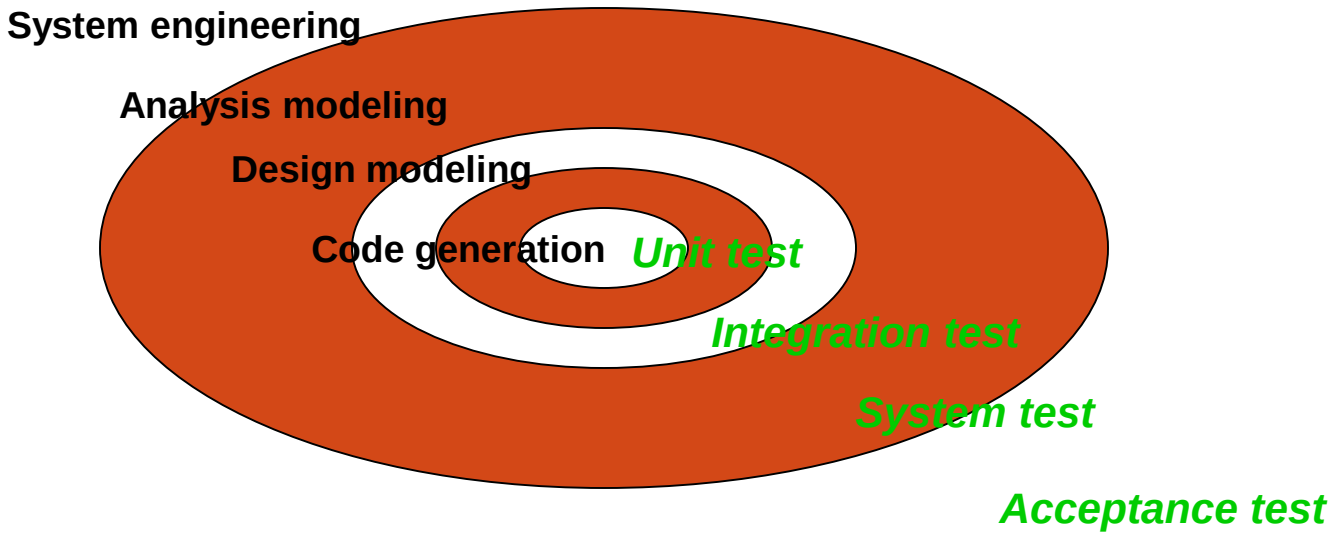
but, will attempt to break it

and, is driven by quality

Good Testing Practices

- A good test case is one that has a high probability of detecting an undiscovered defect, not one that shows that the program works correctly
- It is impossible to test your own program
- A necessary part of every test case is a description of the expected result

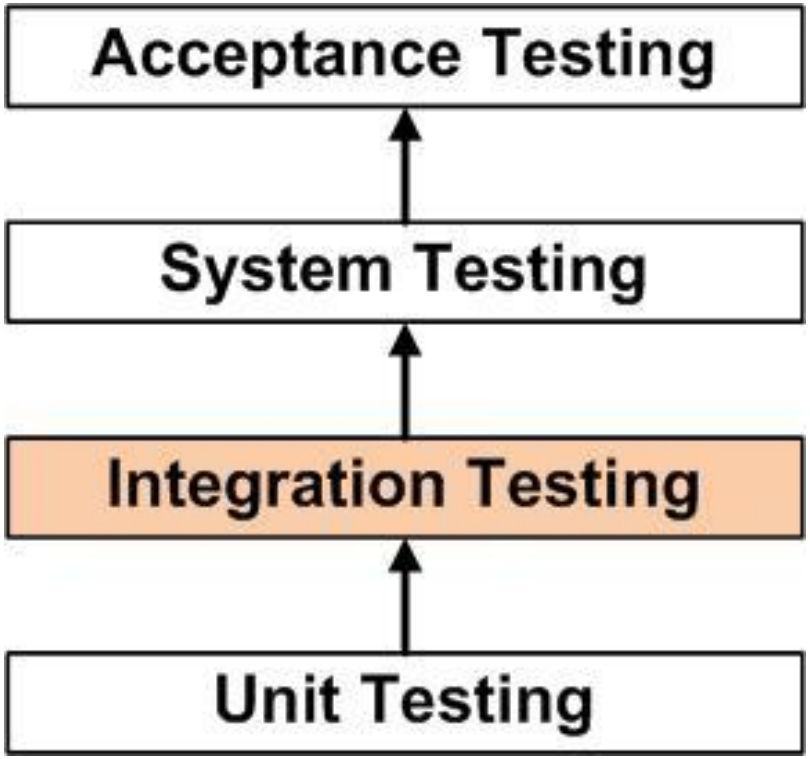
Testing Strategy



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

LEVELS OF TESTING

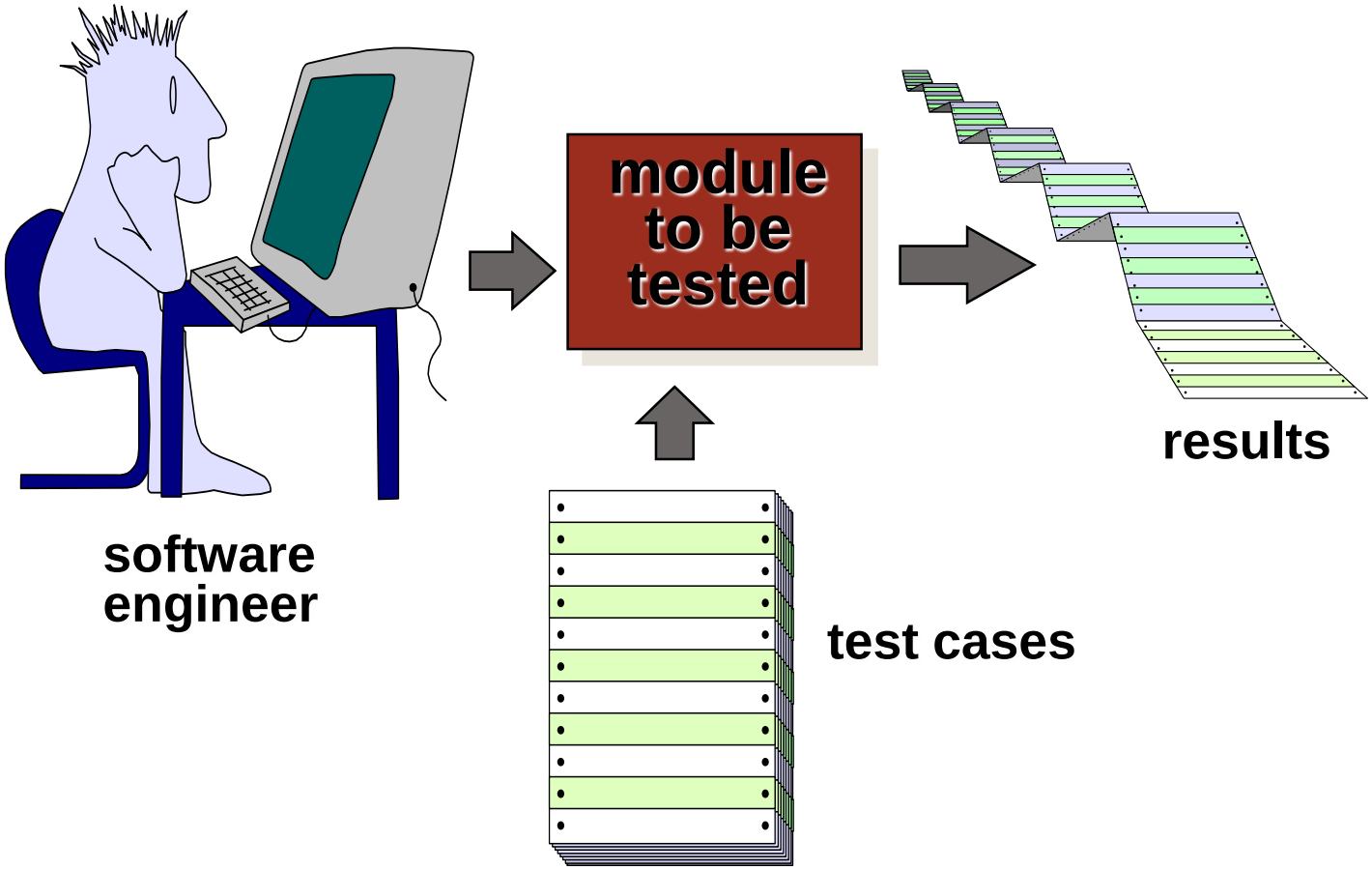


Stubs & Drivers

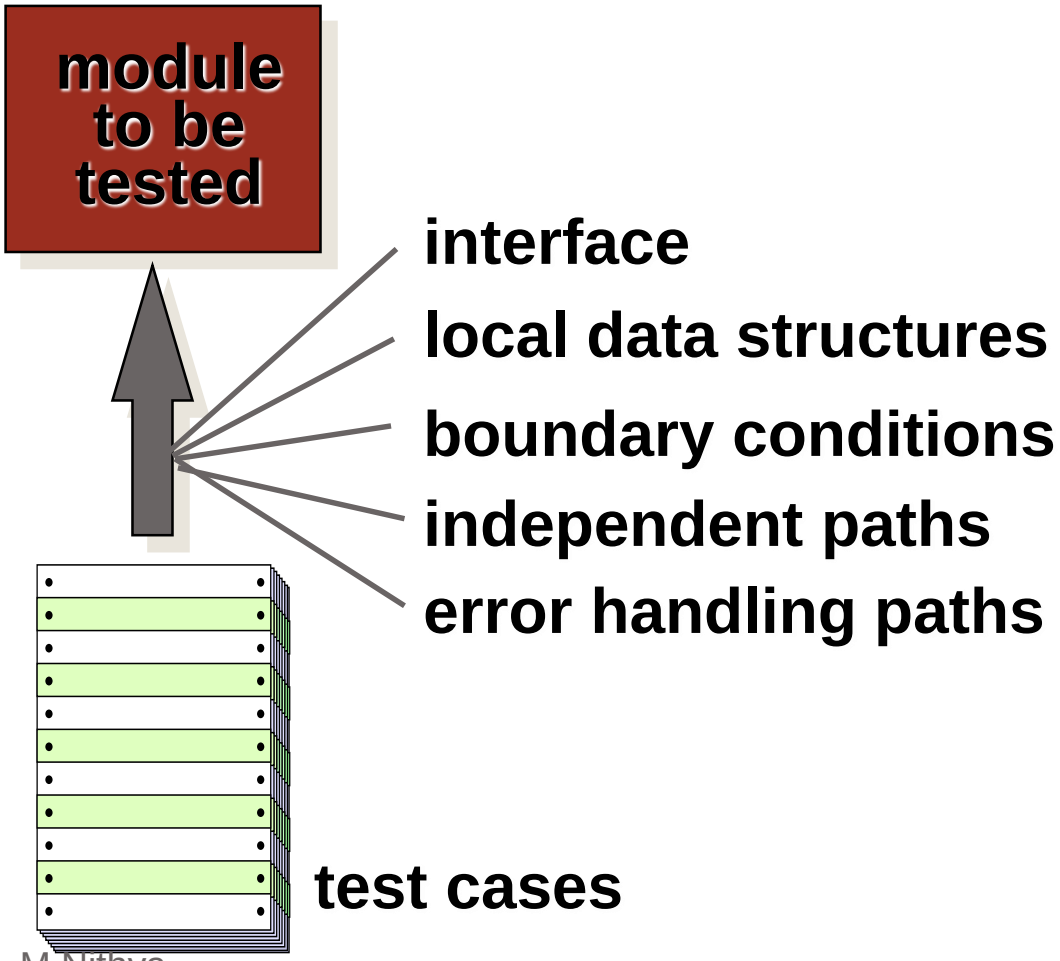
- **Stubs and Drivers** are computer programs which act as a substitutes of some other modules which are not available for testing.

- These computer programs will simulate the functionalities of the other modules thereby facilitating the software testing activity.

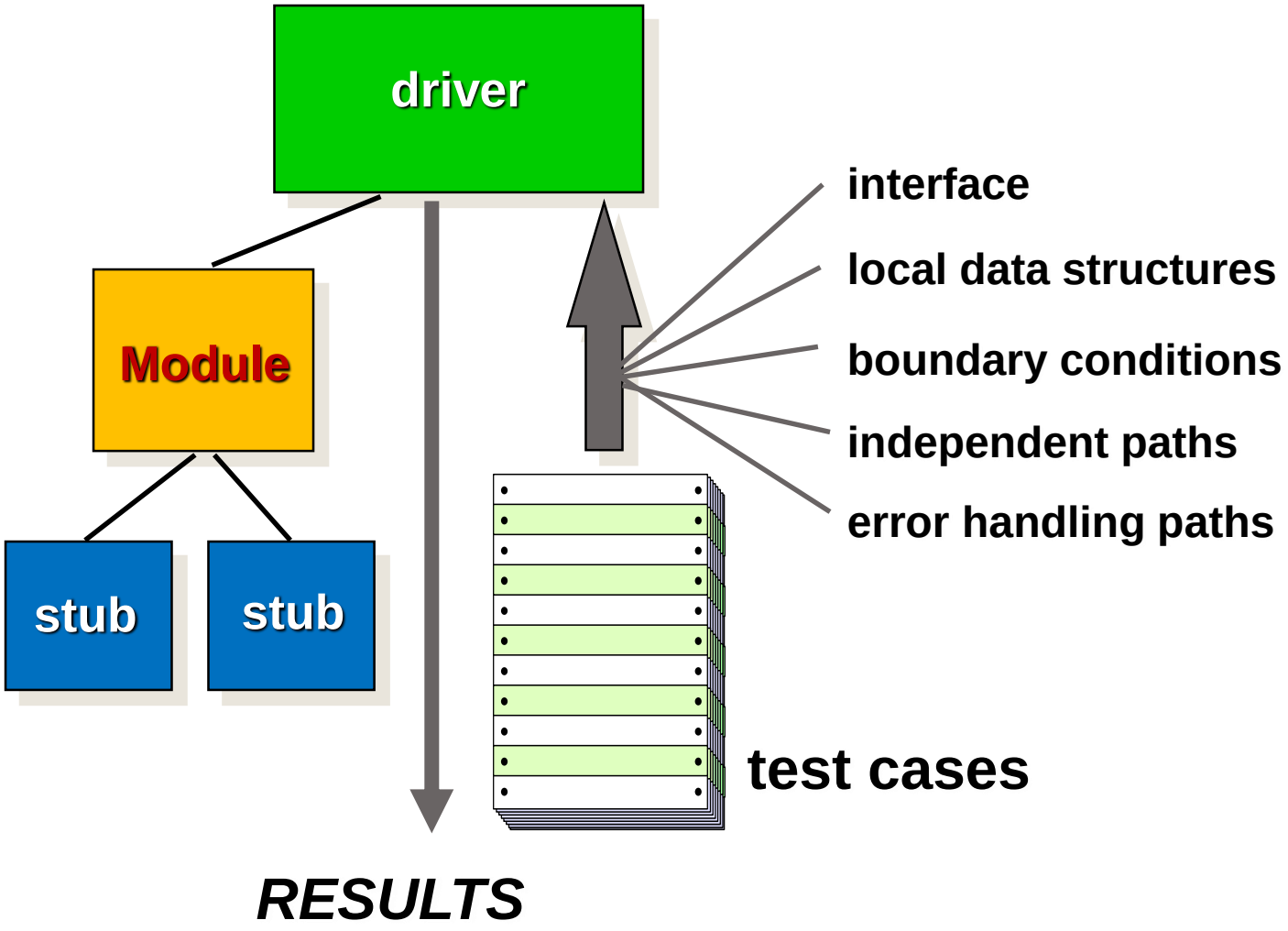
1.Unit Testing



Unit Testing



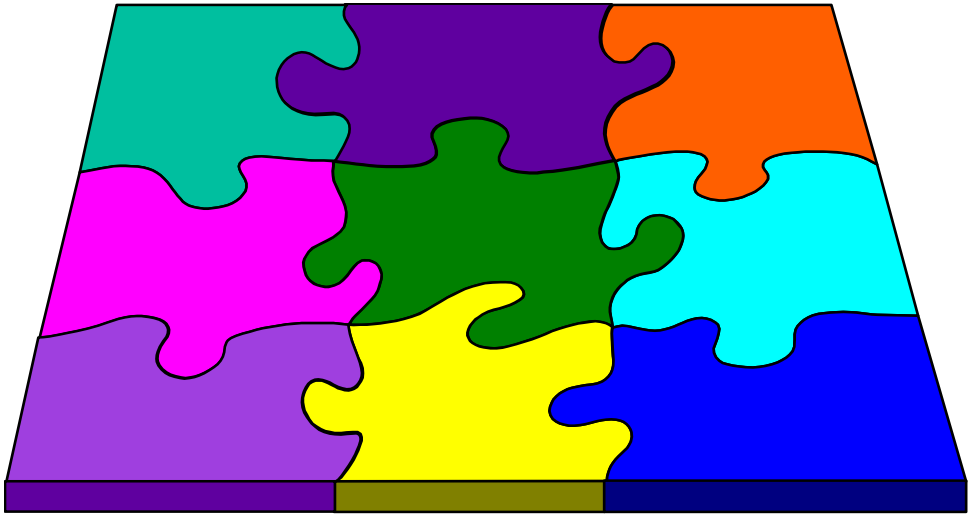
Unit Test Environment



2.Integration Testing Strategies

Options:

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



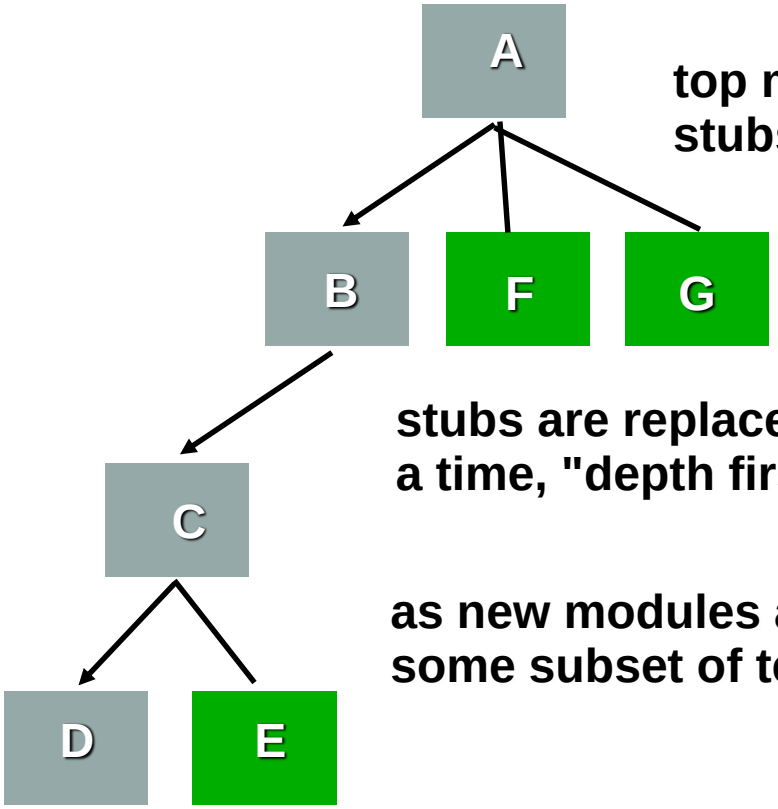
Why Integration Testing Is Necessary

- One module can have an adverse effect on another
- Sub-functions, when combined, may not produce the desired major function
- Individually acceptable imprecision in calculations may be magnified to unacceptable levels
- Interfacing errors not detected in unit testing may appear
- Timing problems (in real-time systems) are not detectable by unit testing
- Resource contention problems are not detectable by unit testing

Top-Down Integration

1. The main control module is used as a driver, and stubs are substituted for all modules directly subordinate to the main module.
2. Depending on the integration approach selected (depth or breadth first), subordinate stubs are replaced by modules one at a time.
3. Tests are run as each individual module is integrated.
4. On the successful completion of a set of tests, another stub is replaced with a real module
5. Regression testing is performed to ensure that errors have not developed as result of integrating new modules

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

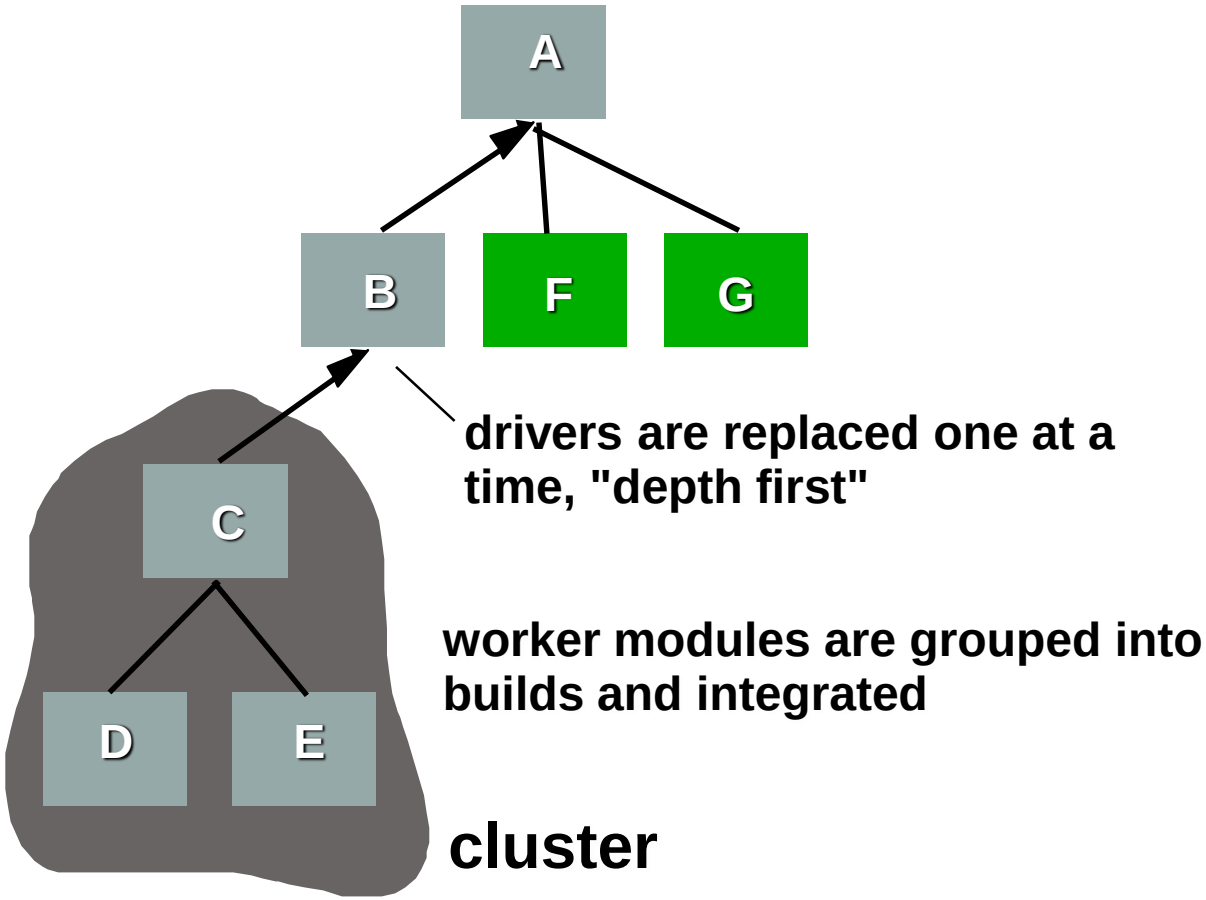
Problems with Top-Down Integration

- Many times, calculations are performed in the modules at the bottom of the hierarchy
- Stubs typically do not pass data up to the higher modules
- Delaying testing until lower-level modules are ready usually results in integrating many modules at the same time rather than one at a time
- Developing stubs that can pass data up is almost as much work as developing the actual module

Bottom-Up Integration

- Integration begins with the lowest-level modules, which are combined into clusters, or builds, that perform a specific software sub-function
- Drivers (control programs developed as stubs) are 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

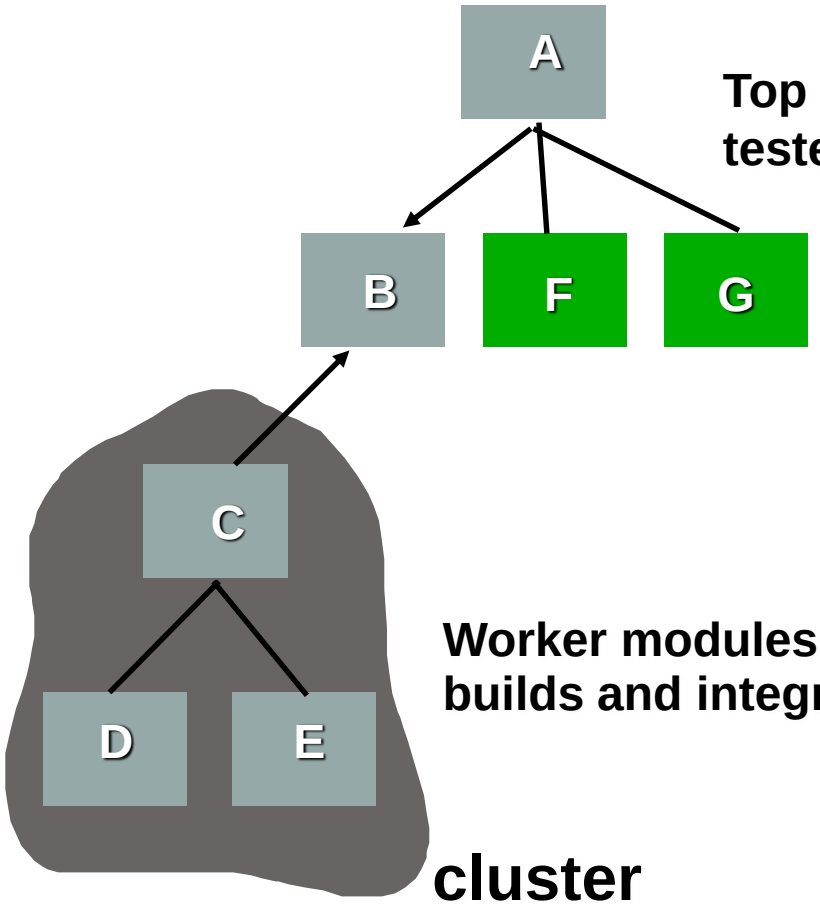
Bottom-Up Integration



Problems with Bottom-Up Integration

- The whole program does not exist until the last module is integrated
- Timing and resource contention problems are not found until late in the process

Sandwich Testing



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

cluster

3) System testing:

- System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.
- System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

4) Acceptance testing:

- Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery.
- Acceptance testing is basically done by the user or customer.
- However, other stackholders can be involved in this process.
  - Alpha testing
  - Beta testing
- Alpha testing is done on the side of developers. This is done at the end of the development process.
- Beta testing, beta testing is carried out on the customer side. This is done just before the launch of the product.

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

- 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.

High Order Testing

- Validation testing
  - Focus is on software requirements
- System testing
  - Focus is on system integration
- 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