Chapter 1

Software Testing Strategies

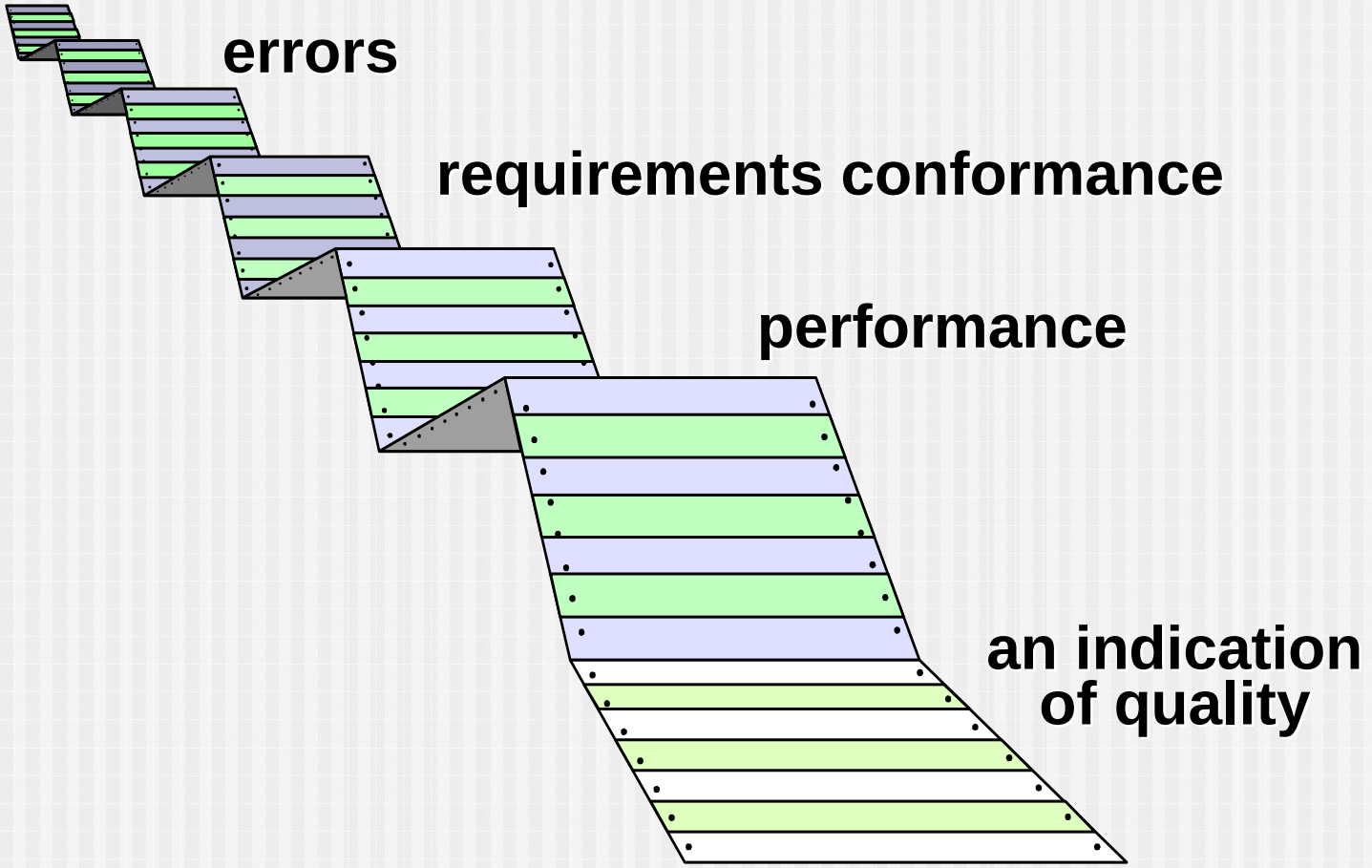
Note : These slides are modified version of slides of Roger Pressman, 6th edition.

Prepared by Narayan D. G., Faculty, BVBCET Hubli.

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



Strategic Approach

General Characteristics of Testing Strategies

- 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.

Testing Strategy: V & V

- *Verification* refers to the set of tasks that ensure that software conforms to its specification
  - Typically involves reviews and meeting to evaluate documents, plans, code, requirements, and specifications. This can be done with checklists, issues lists, walkthroughs, and inspection meeting.

- *Validation* refers to a different set of tasks that ensure that the software conforms to actual customer requirements.
  - typically involves actual testing

  - *Verification:* "Are we building the product right?"
  - *Validation:* "Are we building the right product?“

Who Tests the Software?

developer

Understands the system

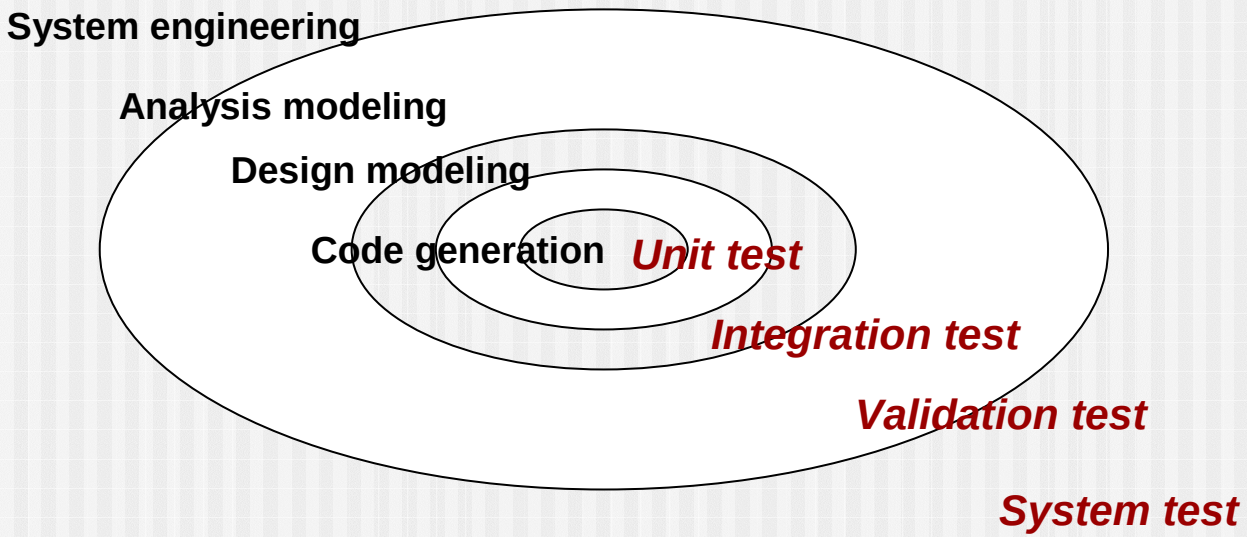
but, will test "gently"

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



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

Strategic Issues

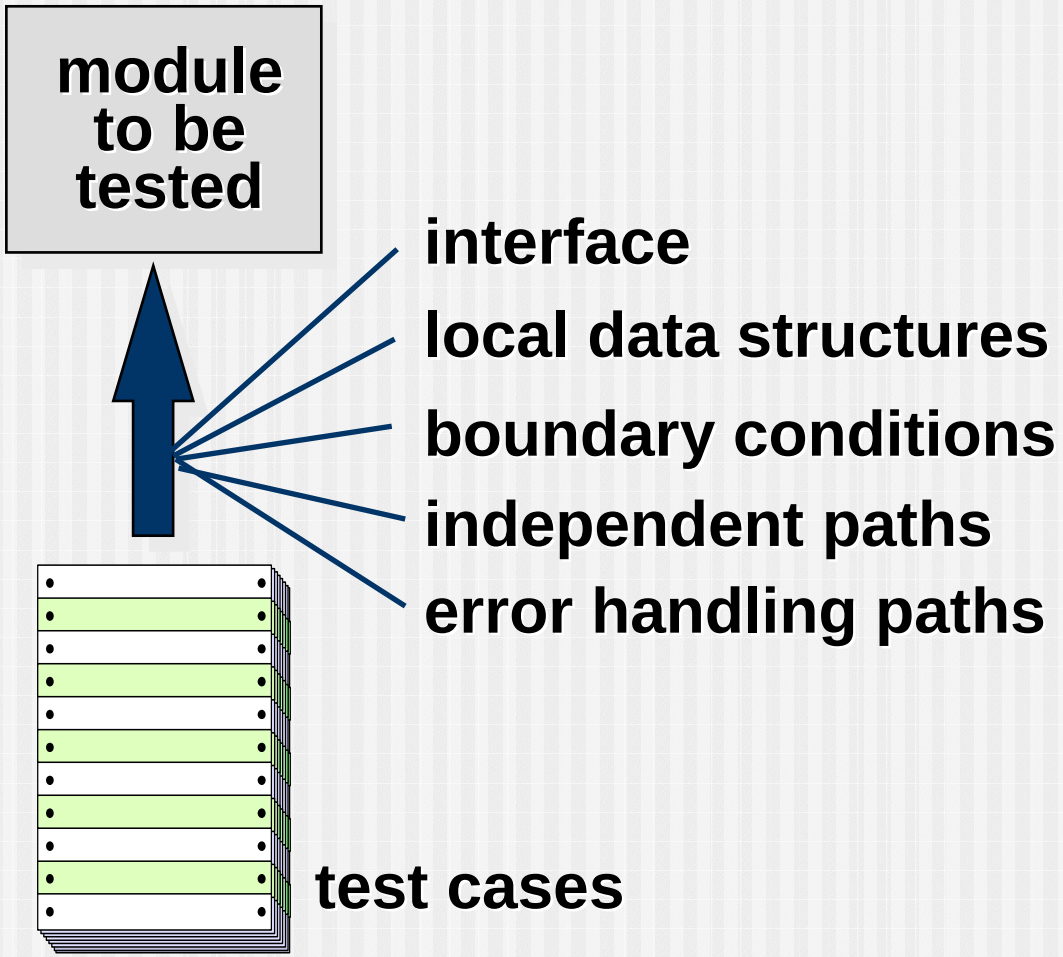
Strategic issues that need to be addressed for successful implementation of testing strategy.

- Specify product requirements in a quantifiable manner long before testing commences.
- 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 technical reviews as a filter prior to testing
- Conduct technical reviews to assess the test strategy and test cases themselves.

Unit Testing

- What errors are commonly found during testing of your programs ?? (Not syntax errors!!)

Unit Testing



Unit Testing

1. Interface test

Module interface is tested to ensure that information properly flows into and out of the program unit.

2. Local Data Structure

Data stored temporarily maintains its integrity during all steps in execution. Also, Local impact of global data can be checked.

3. Boundary Conditions

Error often occur at the boundaries. Example: loop start and end.

Unit Testing

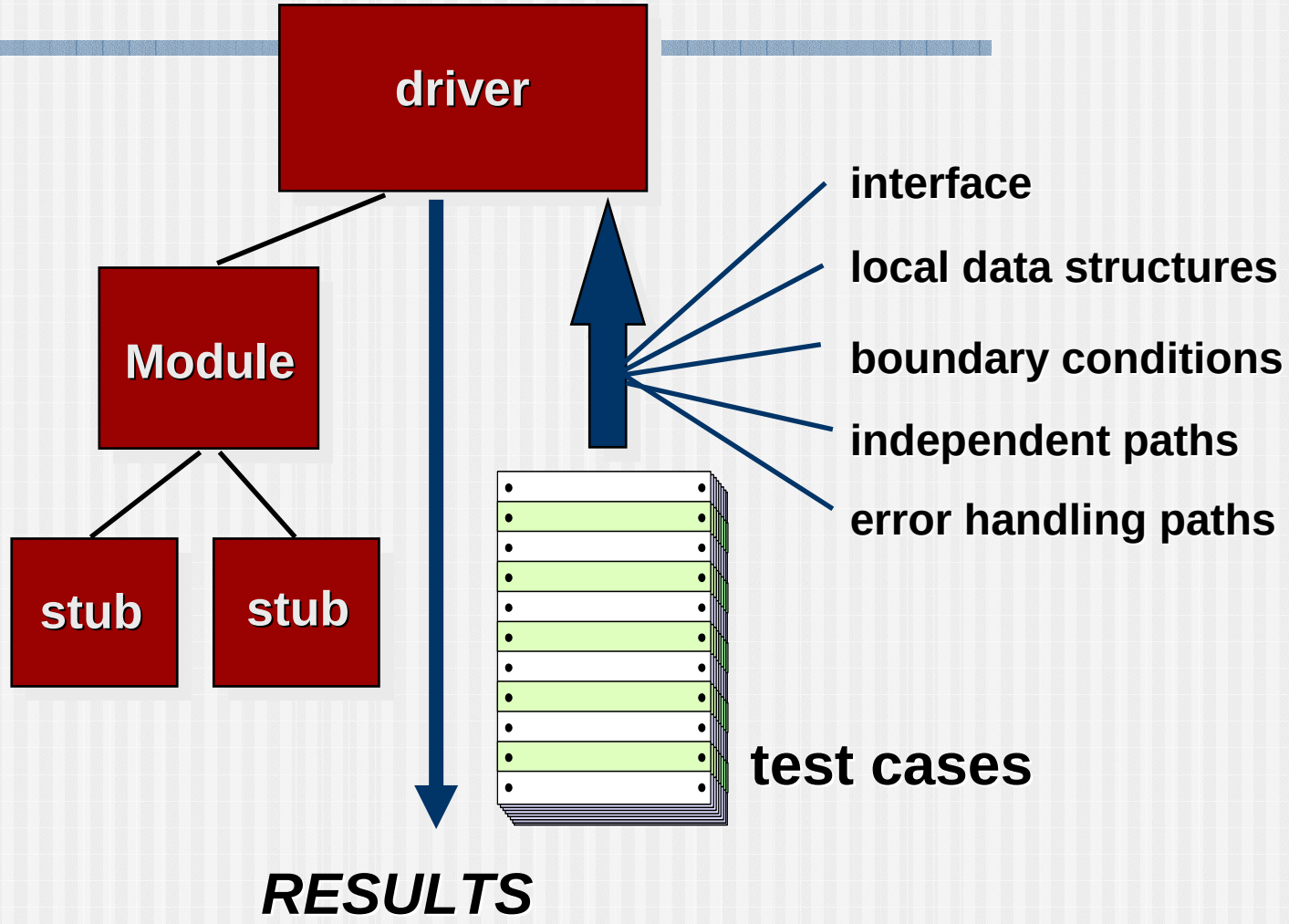
- **Independent paths**

  All the independent path should be exercised to ensure that all the statements in a module have executed at least once.

- **Error Handling Paths**

  Good design dictates that error conditions be anticipated. So, error handling path are set up to reroute or cleanly terminate the processing. (called as antibugging).

Unit Test Environment



Unit Test Environment

- To conduct unit testing drivers and stubs are required.
- Drivers are like main programs which accept test case data, passes such data to module and prints results
- Stubs are dummy subprograms that are used as subordinate to the module to be tested. Stubs may do minimal manipulation, provides verification of entry or sometimes simulate simple functionality.
- Drivers and stubs represent overhead. Both are written but not delivered with final s/w. In some cases, complete testing can be postponed until the integration step.

Integration Testing

- “If they work individually, why do you doubt if they will work when you put them together”
- Problem is with interfacing. Data can be lost across interface; subfunctions, when combined may not produce the desired major function; global data structure can present problems.
- **Definition:** It is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interface.

Integration Testing Strategies

Two Options:

- **The "big bang" approach**

  The entire program is tested as a whole

  Advantages

  Convenient for small systems

  Disadvantages

  1. Need driver and stubs for each module

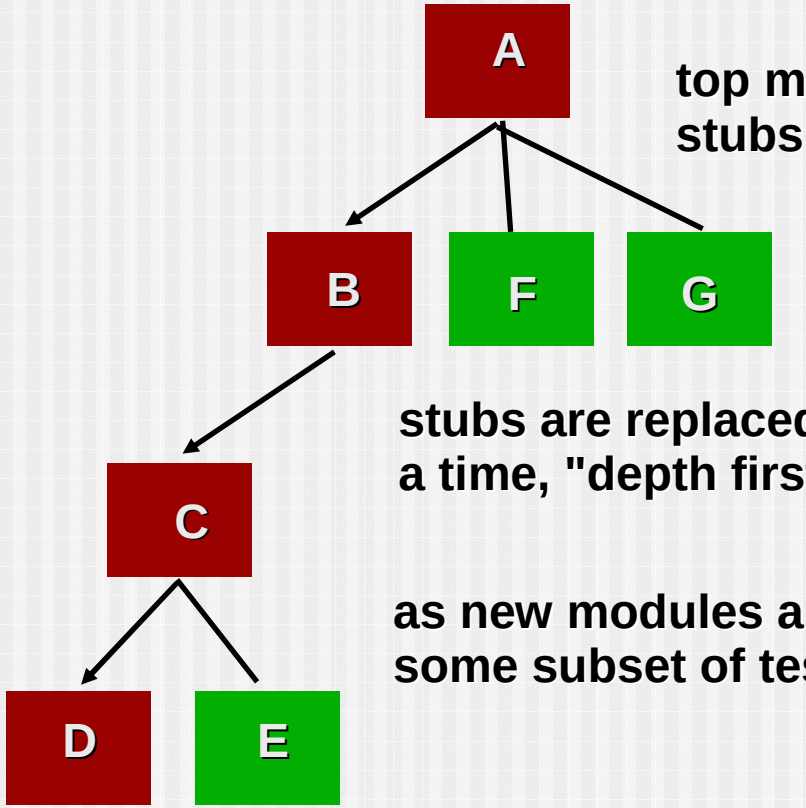
  2. Integration testing can only begin when all modules are ready

  3. Fault localization difficult

  4. Easy to miss interface faults

- **Incremental construction strategy**
  - **Program is constructed and tested in small increments**
  - **Errors are easy to locate**

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

- **Incremental approach to construction of software architecture**
- **The Modules are integrated by moving downward through the control hierarchy in depth-first or breadth-first manner.**
- **Depth-first integration integrates all the components on a major control path of the program structure. Selection of major path is arbitrary and depends on application-specific characteristics.**
- **Breadth-first integration incorporates all components at each level moving across the structure horizontally.**

Top Down Integration

Integration testing is performed in series of five steps.

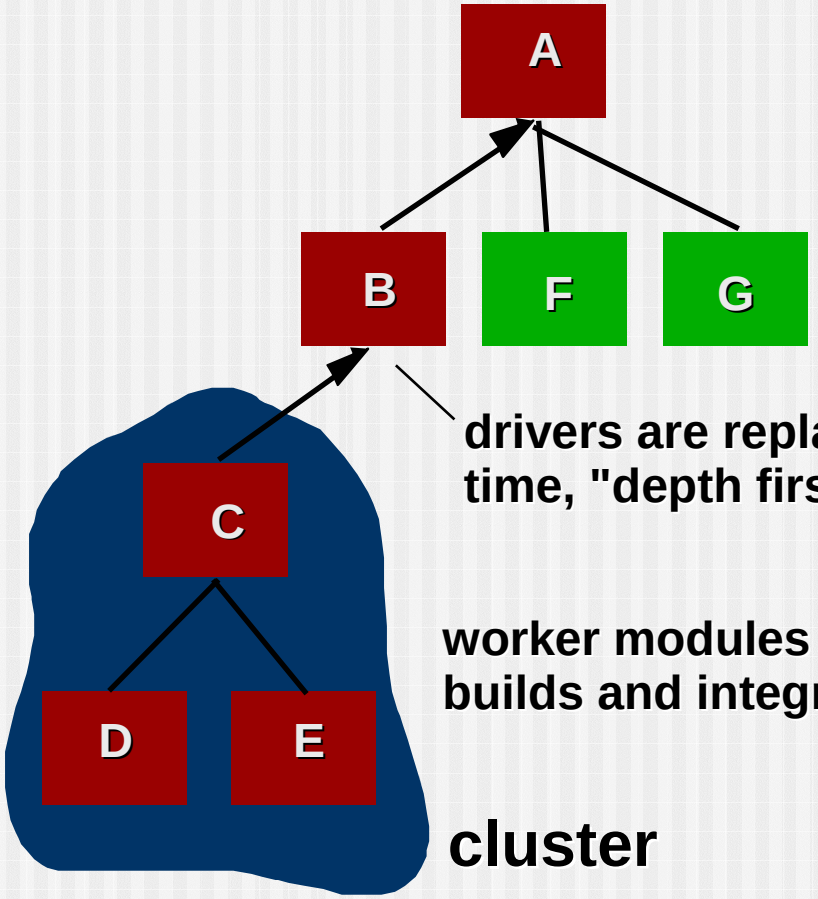
1. **The main control module is used a test driver and stubs are substituted for all components subordinate to the main control module.**
2. **Depending on the integration approach selected, subordinate stubs are replaced one at a time with actual components.**
3. **Tests are conducted as each component is integrated**
4. **On completion of each set of tests, another stub is replaced with the real component**
5. **Regression testing is conducted**

The process continues from step2 until program structure is built

Top Down Integration

- Advantages
  - Fault localization easier
  - Few or no drivers needed
  - Possibility to obtain an early prototype
  - Major design flaws found first
    - in logic modules on top of the hierarchy
- Disadvantages
  - Need lot of stubs / mock objects
  - Potentially reusable modules (in bottom of the hierarchy) can be inadequately tested

Bottom-Up Integration



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

worker modules are grouped into builds and integrated

cluster

Bottom-Up Integration

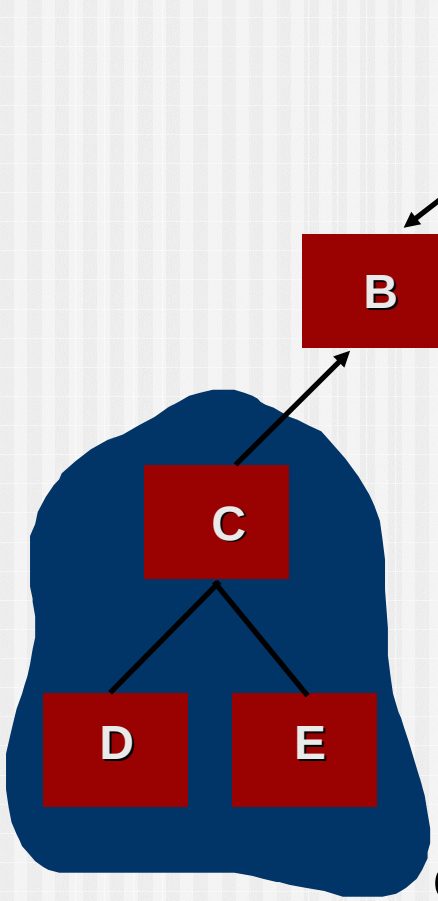
Steps involved in Bottom-up Testing

- Low level components are combined into clusters to perform software subfunction
- A driver 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.

Bottom-Up Integration

- Advantages
  - Fault localization easier (than big-bang)
  - No need for stubs
  - Logic modules tested thoroughly
  - Testing can be in parallel with implementation
- Dis-advantages
  - Need drivers
  - High-level modules (that relate to the solution logic) tested in the last (and least)
  - No concept of early skeletal system

Sandwich Testing



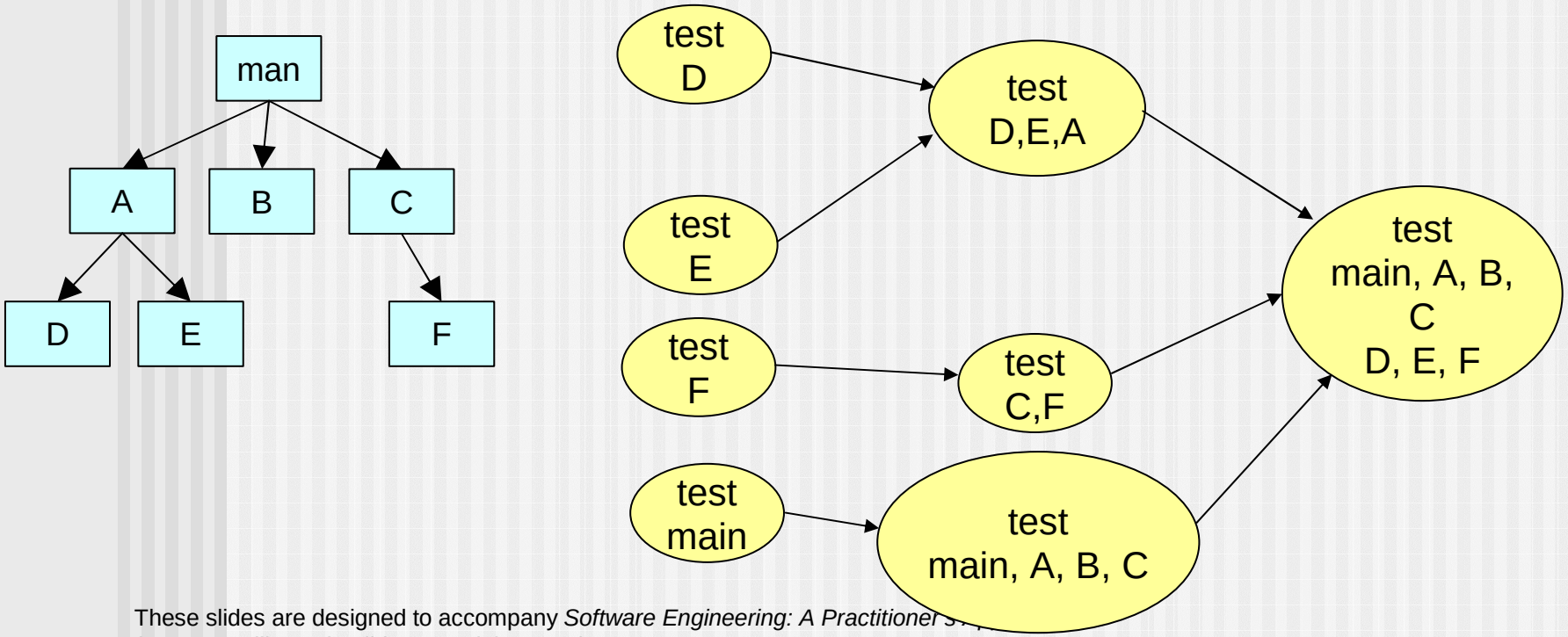
Top modules are tested with stubs

Worker modules are grouped into builds and integrated

cluster

Sandwich Integration

- Combines top-down and bottom-up approaches
- Distinguish 3 layers
  - logic (top) - tested top-down
  - middle
  - operational (bottom) – tested bottom-up



Regression Testing

- Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed.
- *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
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.

Regression Testing

- Regression test suite contains the 3 different classes of test cases
  - A representative sample of tests that will exercise all s/w functions
  - Additional tests that focus on software functions that are likely to be affected by change
  - Tests that focus on software components that have been changed.
- As integration testing proceeds, the number of regression tests can grow.

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.

Smoke Testing

Benefits of smoke testing when it is applied on complex, time critical projects

- Integration risk is minimized

  smoke testing conducted daily. So, show-stopper errors are uncovered early

- Quality of the end product is improved

  Uncovers both functional, architectural and component-level errors.

- Error diagnosis and correction are simplified.

- Progress is easier to asses.

Integration Test documentation

- Describes test plan
- Test phases for software, what type of tests need to be carried out, schedule, resources required, test case specification and test report.
- Example, CAD system, the phases are
  - User interaction (command selection, drawing creation etc)
  - Data manipulation (symbol creation rotation etc)
  - Display processing (2-D and 3-D display, graphs, charts etc)
  - Database management (access, update, performance etc)

  Type of tests may be to check interface integrity, functional integrity, information content and performance.

Object-Oriented Testing

- begins by evaluating the correctness and consistency of the analysis and design models
- testing strategy

  Class Testing

  Integration of classes

  Validation testing

  System Testing

Object Class Testing

- It’s a unit testing in OO context
- Complete test coverage of a class involves
  - Testing all operations associated with an object
  - Setting and interrogating all object attributes
  - Exercising the object in all possible states
- Inheritance makes it more difficult to design object class tests as the information to be tested is not localised

IntegrationTesting Strategy

- integration applied 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
    - Has the advantage that it tests system features as experienced by users
  - cluster testing—integrates the set of classes required to demonstrate one collaboration.

Validation and Acceptance testing

Validation Testing

- Ensure that each function or performance characteristic conforms to its specification.
- Deviations (deficiencies) must be negotiated with the customer to establish a means for resolving the errors.

Acceptance Testing

- Making sure the software works correctly for intended user in his or her normal work environment.
- For Generic Products the following two types of test are carried out.

Alpha test

version of the complete software is tested by customer under the supervision of the developer at the developer's site

Beta test

version of the complete software is tested by customer at his or her own site without the developer being present

System testing

- It is a series of different steps whose primary purpose is to fully exercise the computer-based system.
- Focus is on system integration.

- Recovery testing
  - System should be fault tolerant or system failure should be corrected within specified time.
  - forces the software to fail in a variety of ways and verifies that recovery is properly performed.
  - Recovery may be automatic or manual.

- Security testing
  - verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration
  - Tester will try to penetrate the system by acquiring passwords, overwhelming the system, cause errors etc.

System testing

- Stress testing
  - Evaluates a system or component at or beyond the limits of its specified requirements.
  - Testing is performed under unusually heavy loads, heavy repetition of certain actions or inputs, input of large numerical values, large complex queries to a database, etc.
- Performance Testing
  - test the run-time performance of software within the context of an integrated system
  - Often coupled with stress testing