Lecture 11

- Software Testing Strategies
- “How do you organize your software tests?”

Topics

- Software Testing Strategy
- Unit Testing
- Integration Testing
- System Testing
- Testing Object-Oriented Applications
- Testing Web Applications
- Testing Mobile Apps

1. Software Testing Strategy

Strategic Approach

- To perform effective testing, you should **conduct effective technical review**s. 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.

Verification & Validation

- *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?

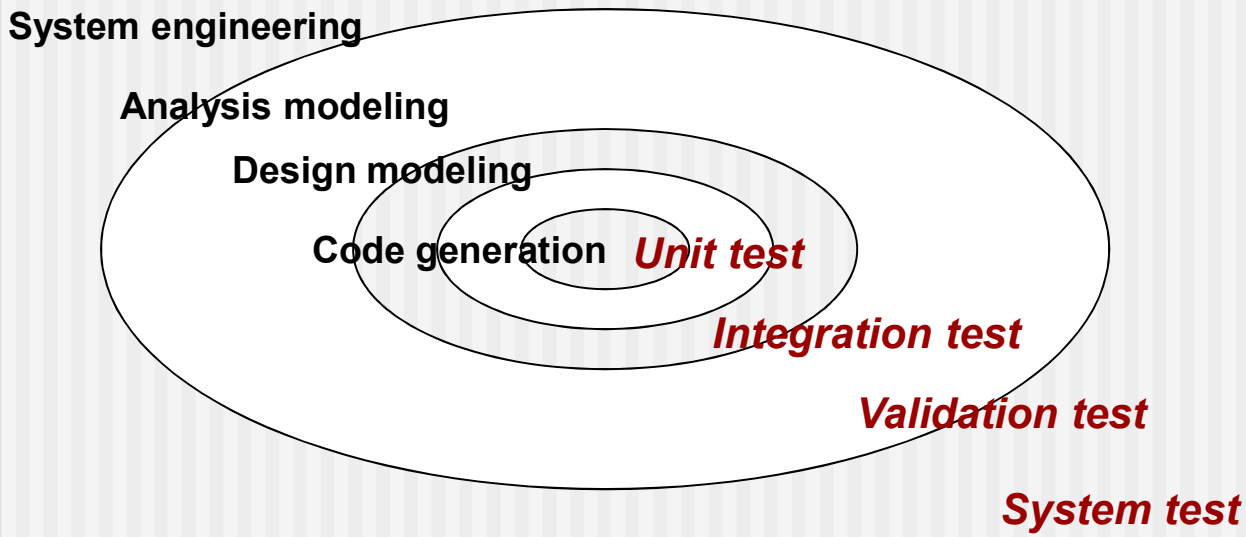
Developer

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



Overall Testing Strategies

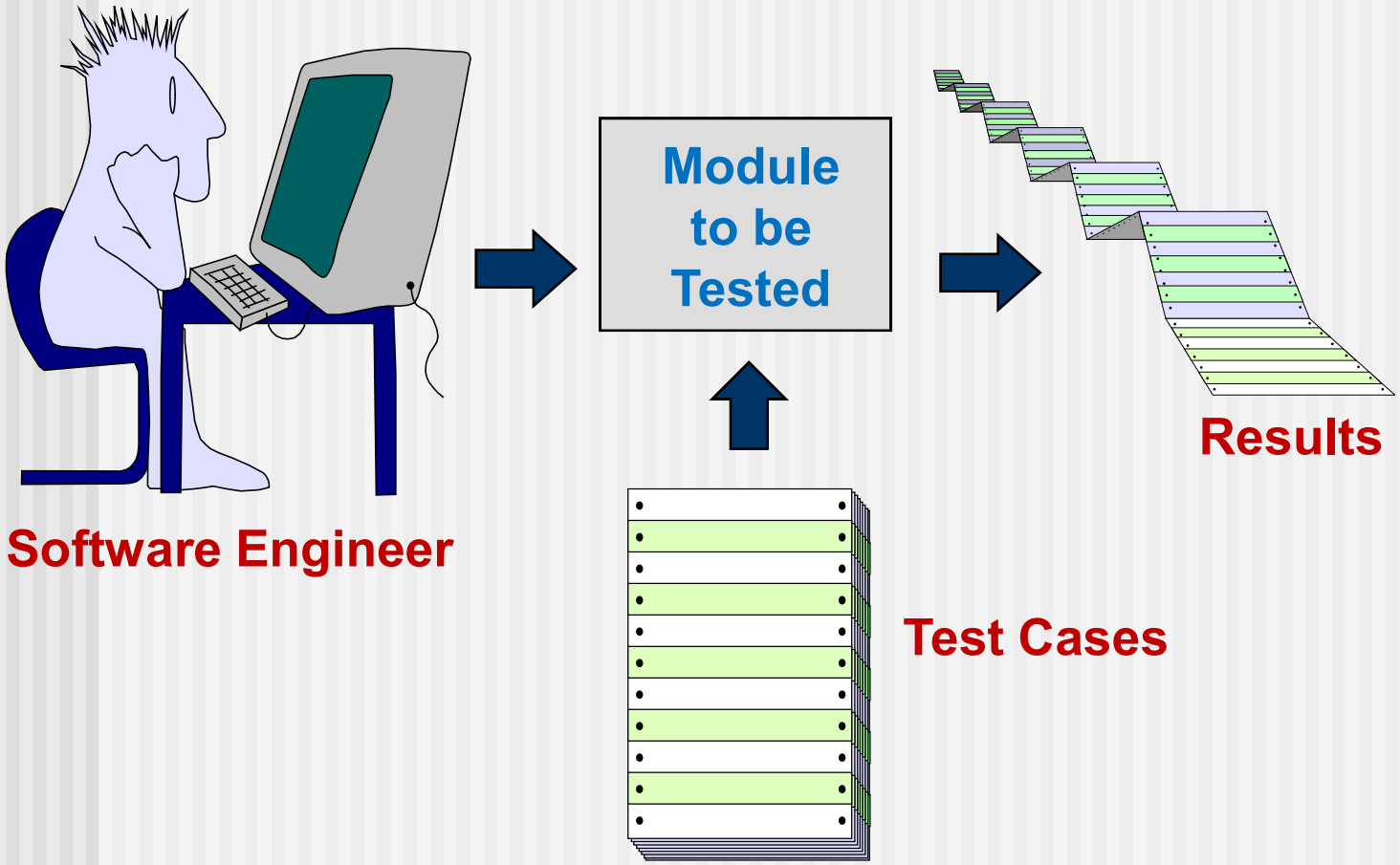
- We begin by **'testing-in-the-small'** and move toward **'testing-in-the-large'**.
- For conventional software:
  - The module (component) is our initial focus.
  - Integration of modules follows.
- For OO software:
  - Our focus when "testing in the small" changes from an individual module (the conventional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

Strategic Issues

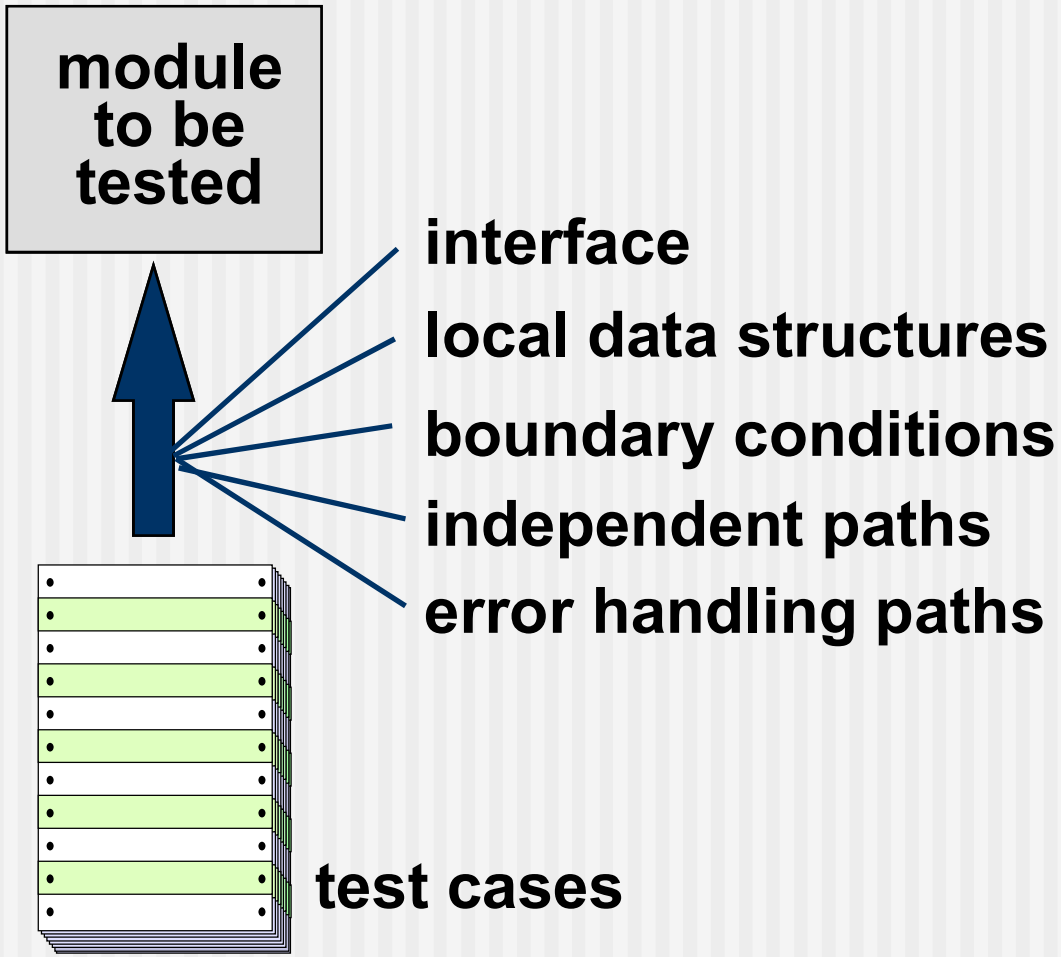
- Specify product requirements in a quantifiable manner long before testing commences.
- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes “rapid cycle testing.”
- Build “robust” software 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.
- Develop a continuous improvement approach for the testing process.

2. Unit Testing

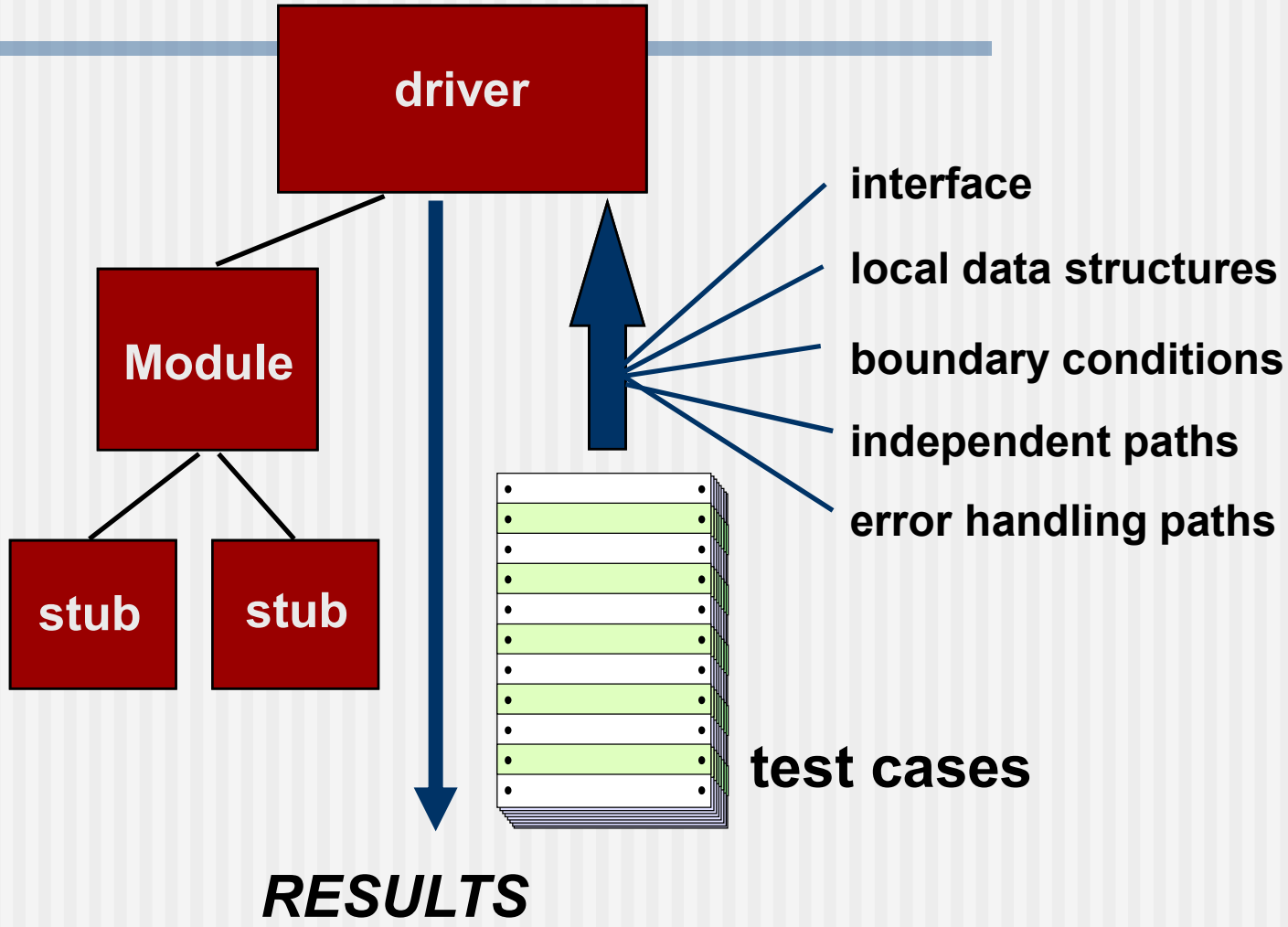
Unit Testing



Unit Testing



Unit Test Environment

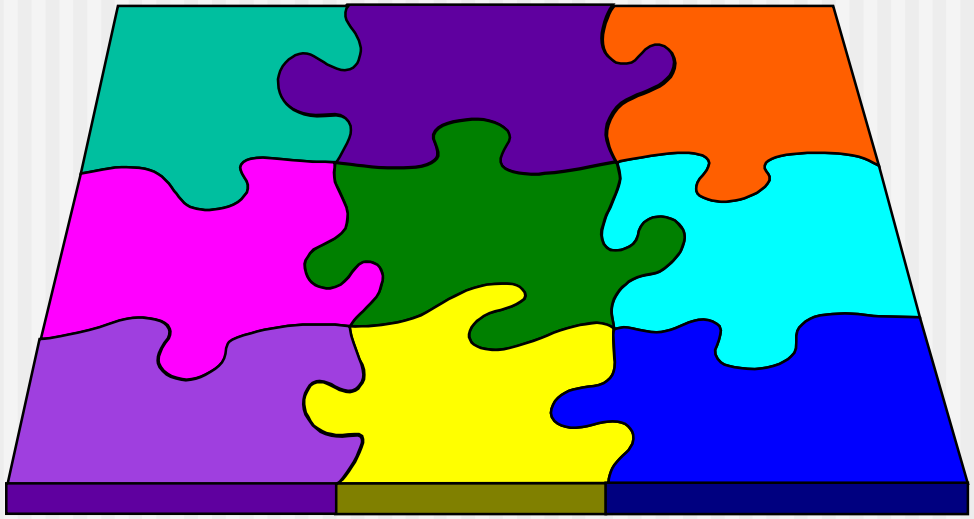


3. Integration Testing

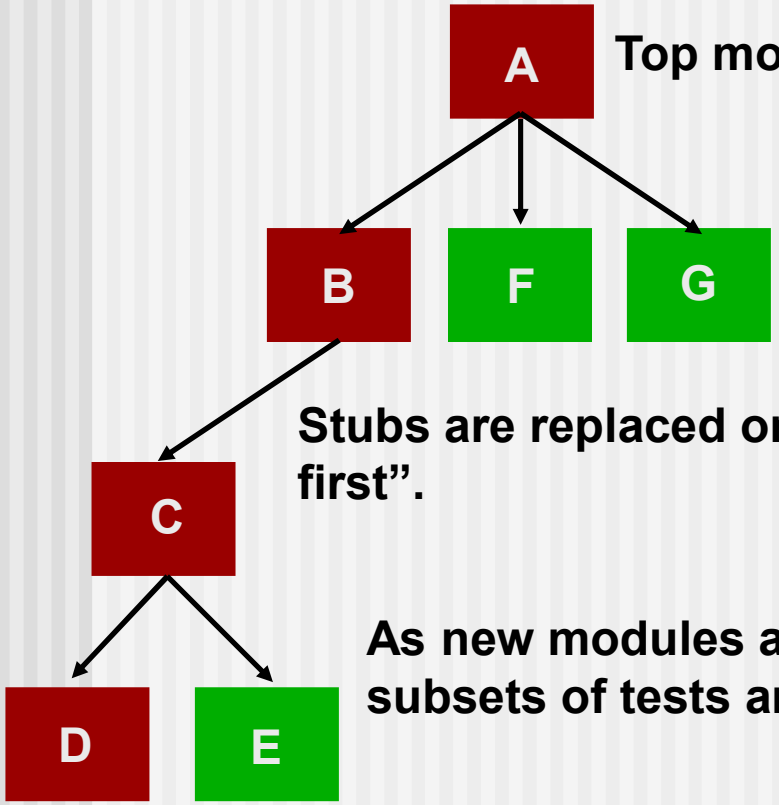
Integration Testing Strategies

Options:

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



Top Down Integration

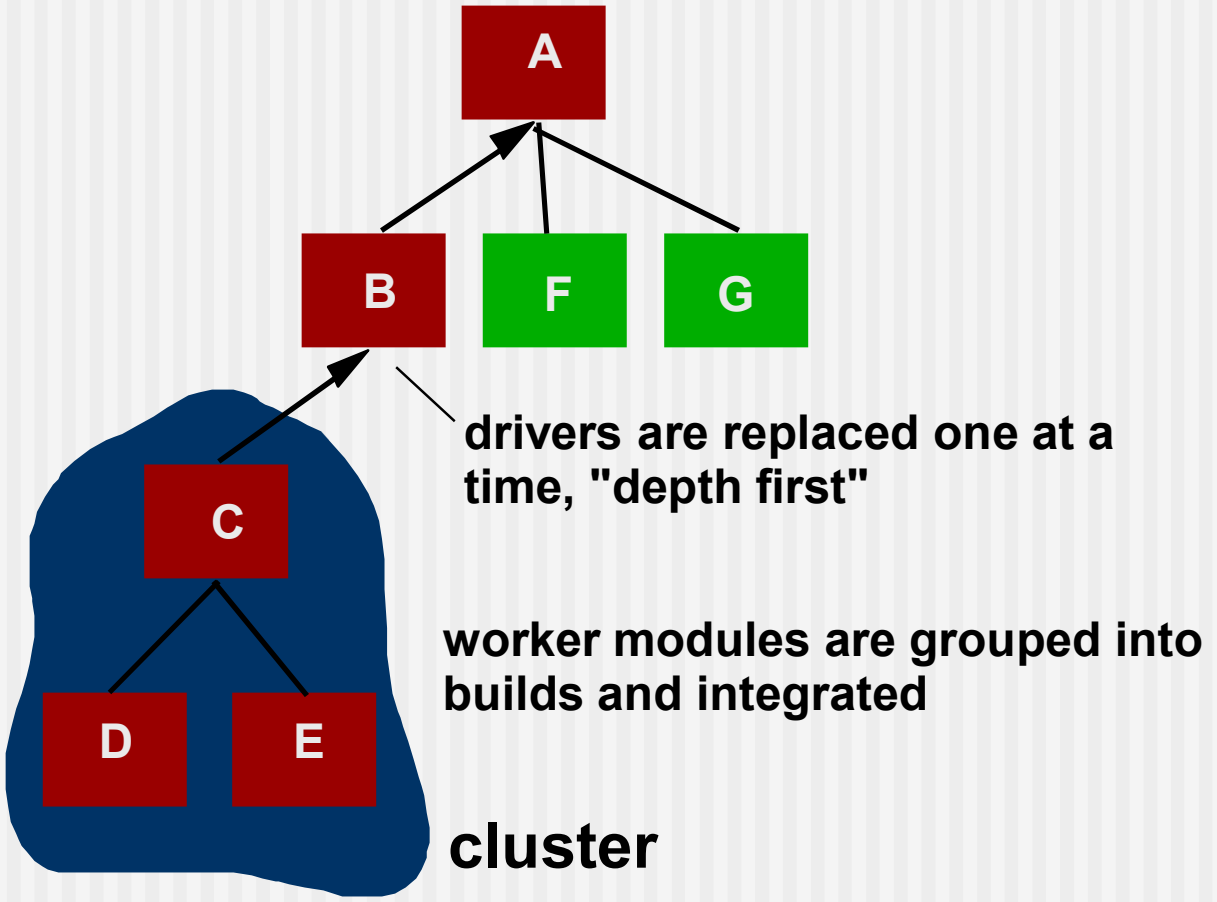


Top module is tested with stubs.

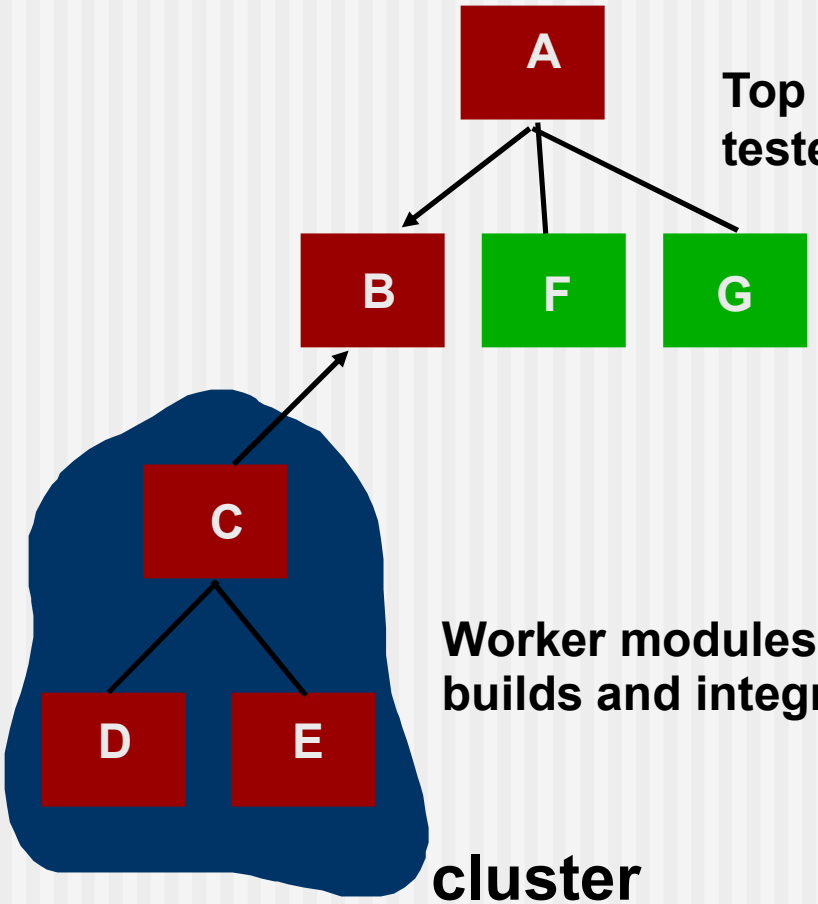
Stubs are replaced one at a time, "depth first".

As new modules are integrated, some subsets of tests are re-run.

Bottom-Up Integration



Sandwich Testing



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

cluster

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.

General Testing Criteria

- Interface integrity – internal and external module interfaces are tested as each module or cluster is added to the software
- Functional validity – test to uncover functional defects in the software
- Information content – test for errors in local or global data structures
- Performance – verify specified performance bounds are tested

4. System Testing

Validation Testing

- Focus on user-visible actions and user-recognizable output from the system
- Validation succeeds when software functions as expected by the customer
- Achieved through a series of tests that demonstrates conformity with requirements
- Includes validation of functional, behavioral and performance requirements, delivery of contents and documentation, and other specified requirements at the beginning of the project

Alpha and Beta Testing

- Alpha Test
  - Conducted at the developer's site by a representative group of end users
  - The software is used in a natural setting with the developer "looking over the shoulder" of the users and recording errors and usage problems
- Beta Test
  - Conducted at one or more end user sites
  - The developer is generally not present – a "live" application of the software in an environment that cannot be controlled by the developer
  - User records all problems and reports to the developer

Acceptance Testing

- Customer acceptance testing is a formal version of beta testing, for custom software that is delivered to customer under contract
- The customer performs a series of specific tests to uncover errors and verify conformity to requirements before accepting the software
- In some cases (e.g. major corporate or governmental systems) the acceptance testing can be very formal and encompass may days or even weeks of testing

High Order Testing

- **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** — Tests the run-time performance of software within the context of an integrated system.

5. Testing Object-Oriented Applications

Object-Oriented Testing

- begins by evaluating the correctness and consistency of the analysis and design 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

OO Testing Strategy

- class testing is the equivalent of unit testing
  - operations within the class are tested
  - the state behavior of the class is examined
- 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
  - cluster testing—integrates the set of classes required to demonstrate one collaboration

6. Testing Web Applications

Testing Quality Dimensions-I

- *Content* is evaluated at both a syntactic and semantic level.
  - syntactic level—spelling, punctuation and grammar are assessed for text-based documents.
  - semantic level—correctness (of information presented), consistency (across the entire content object and related objects) and lack of ambiguity are all assessed.
- *Function* is tested for correctness, instability, and general conformance to appropriate implementation standards (e.g.,Java or XML language standards).
- *Structure* is assessed to ensure that it
  - properly delivers WebApp content and function
  - is extensible
  - can be supported as new content or functionality is added.

Testing Quality Dimensions-II

- *Usability* is tested to ensure that each category of user
  - is supported by the interface
  - can learn and apply all required navigation syntax and semantics
- *Navigability* is tested to ensure that
  - all navigation syntax and semantics are exercised to uncover any navigation errors (e.g., dead links, improper links, erroneous links).
- *Performance* is tested under a variety of operating conditions, configurations, and loading to ensure that
  - the system is responsive to user interaction
  - the system handles extreme loading without unacceptable operational degradation

Testing Quality Dimensions-III

- *Compatibility* is tested by executing the WebApp in a variety of different host configurations on both the client and server sides.
  - The intent is to find errors that are specific to a unique host configuration.
- *Interoperability* is tested to ensure that the WebApp properly interfaces with other applications and/or databases.
- *Security* is tested by assessing potential vulnerabilities and attempting to exploit each.
  - Any successful penetration attempt is deemed a security failure.

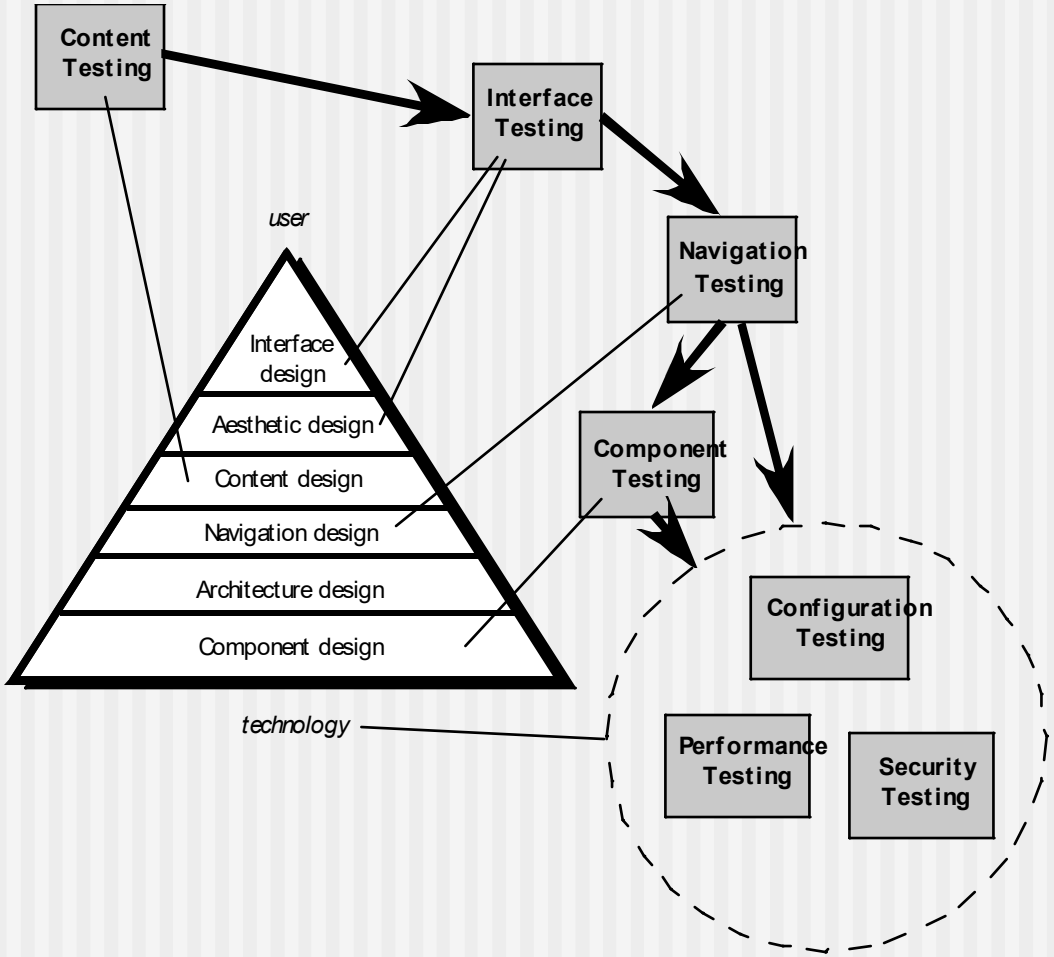
WebApp Testing - I

- The content model for the WebApp is reviewed to uncover errors.
- The interface model is reviewed to ensure that all use cases can be accommodated.
- The design model for the WebApp is reviewed to uncover navigation errors.
- The user interface is tested to uncover errors in presentation and/or navigation mechanics.
- Each functional component is unit tested.

WebApp Testing - II

- Navigation throughout the architecture is tested.
- The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration.
- Security tests are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment.
- Performance tests are conducted.
- The WebApp is tested by a controlled and monitored population of end-users. The results of their interaction with the system are evaluated for content and navigation errors, usability concerns, compatibility concerns, and WebApp reliability and performance.

The Testing Process



7. Testing Mobile Apps

Mobile App Testing

- User experience testing – ensuring app meets stakeholder usability and accessibility expectations
- Device compatibility testing – testing on multiple devices
- Performance testing – testing non-functional requirements
- Connectivity testing – testing ability of app to connect reliably
- Security testing – ensuring app meets stakeholder security expectations
- Testing-in-the-wild – testing app on user devices in actual user environments
- Certification testing – app meets the distribution standards

Mobile Testing Guidelines

- Understand the network landscape and device landscape.
- Conduct testing in uncontrolled real-world test conditions.
- Select the right automation test tool.
- Identify the most critical hardware/ platform combinations to test.
- Check the end-to-end functional flow in all possible platforms at least once.
- Conduct performance, GUI, and compatibility testing using actual devices.
- Measure Mobile App performance under realistic network load conditions.