Software Quality Assurance

CHAPTER 4

SOFTWARE TESTING LIFE CYCLE

DR. JAMIL S. ALAGHA

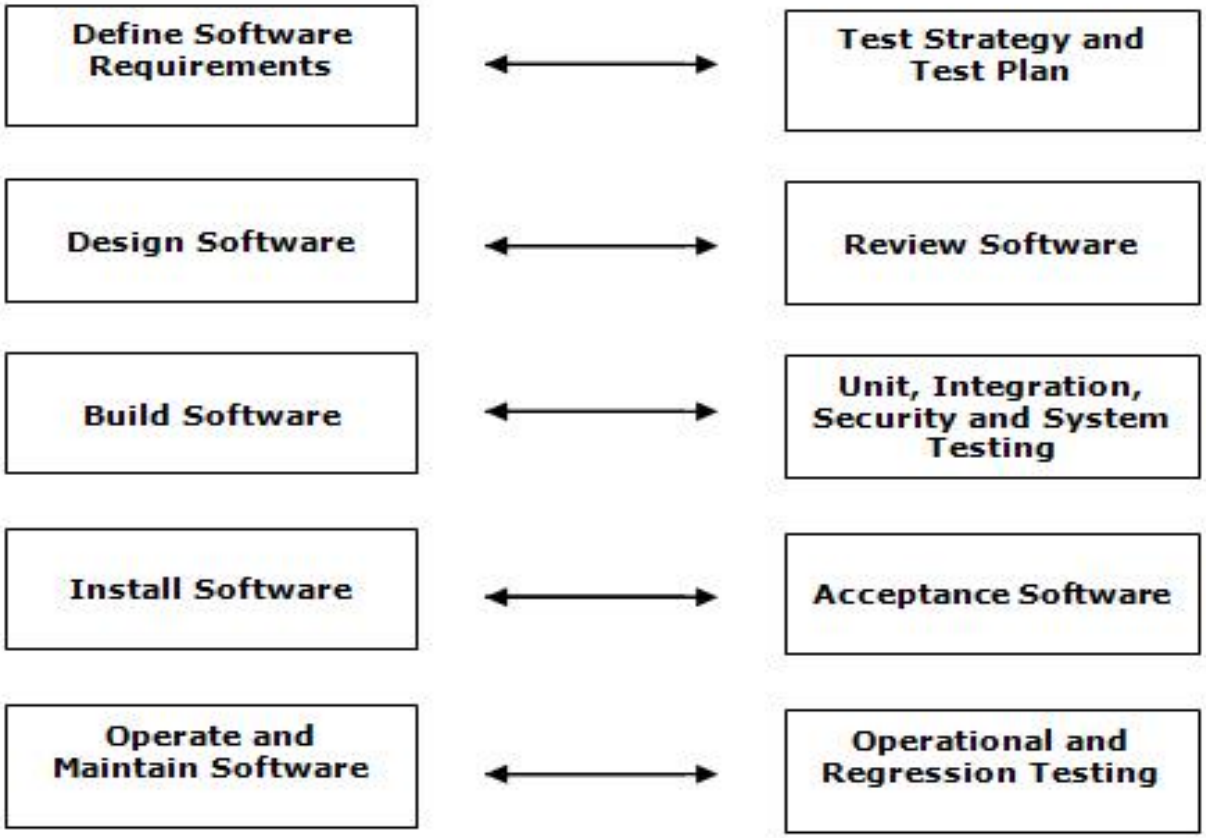
Outline

- **Definition**
- **Test Strategy**
- **Test Plans**
- **Software Review**
- **Unit Testing**
- **Integration Testing**
- **System Testing**
- **Security testing**
- **Acceptance Test**
- **Regression Testing**

Software testing life cycle

Software testing life cycle identifies what test activities to carry out and when (what is the best time) to accomplish those test activities.

Software testing life cycle



Outline

- Definition
- Test Strategy
- Test Plans
- Software Review
- Unit Testing
- Integration Testing
- System Testing
- Security testing
- Acceptance Test
- Regression Testing

Test Strategy

Before we think about making planning for our project, we must first come up with some strategy to deal with risks, probable tasks involved, technology selection, skills matching, required training, automation, tools

Test Strategy

- ***Technological Risks***
- Suppose the application you are going to test involves testing a Web-based Java application as well as testing its integration with a mobile module.
- The mobile module has a component that depends on information about tracking location of the truck using a GPS (Global Positioning System) service.
- What technological factors should be considered risks for testing this mobile module?

Test Strategy

Technological Risks

1. Does a suitable mobile emulator exist which will help in testing this module?

2. Can this mobile emulator be installed successfully on your existing hardware

and operating platform?

3. Does your testing team have prior experience in testing such mobile modules?

4. Does the main application integrate successfully with this mobile emulator?

Test Strategy

Required training

1. Does your team need any training for testing this module?
2. Can you identify the kind of training required?
3. Is this training provided by any training service provider? How much time is needed for training?
4. Include this training time in your project estimates

Test Strategy

Strategy for Automation

Automation is for repeated execution of test cases

Traditionally the ROI for automation works out when any test case has to be executed more than 13 times. After this number of executions, automation works out to be cheaper than manual testing

Test Strategy

Automation Tool Selection

Depending on the kind of project and its requirement, you will have to choose appropriate automation tools.

You need expert automation engineers who can use the tool to give you automated test scripts.

Test Strategy

Other issues

In incremental testing should testing be performed bottom-up or top-down?

Which parts of the testing plan should be performed according to the white box testing model?

Which parts of the testing plan should be performed according to the automated testing model?

Outline

- Definition
- Test Strategy
- Test Plans
- Software Review
- Unit Testing
- Integration Testing
- System Testing
- Security testing
- Acceptance Test
- Regression Testing

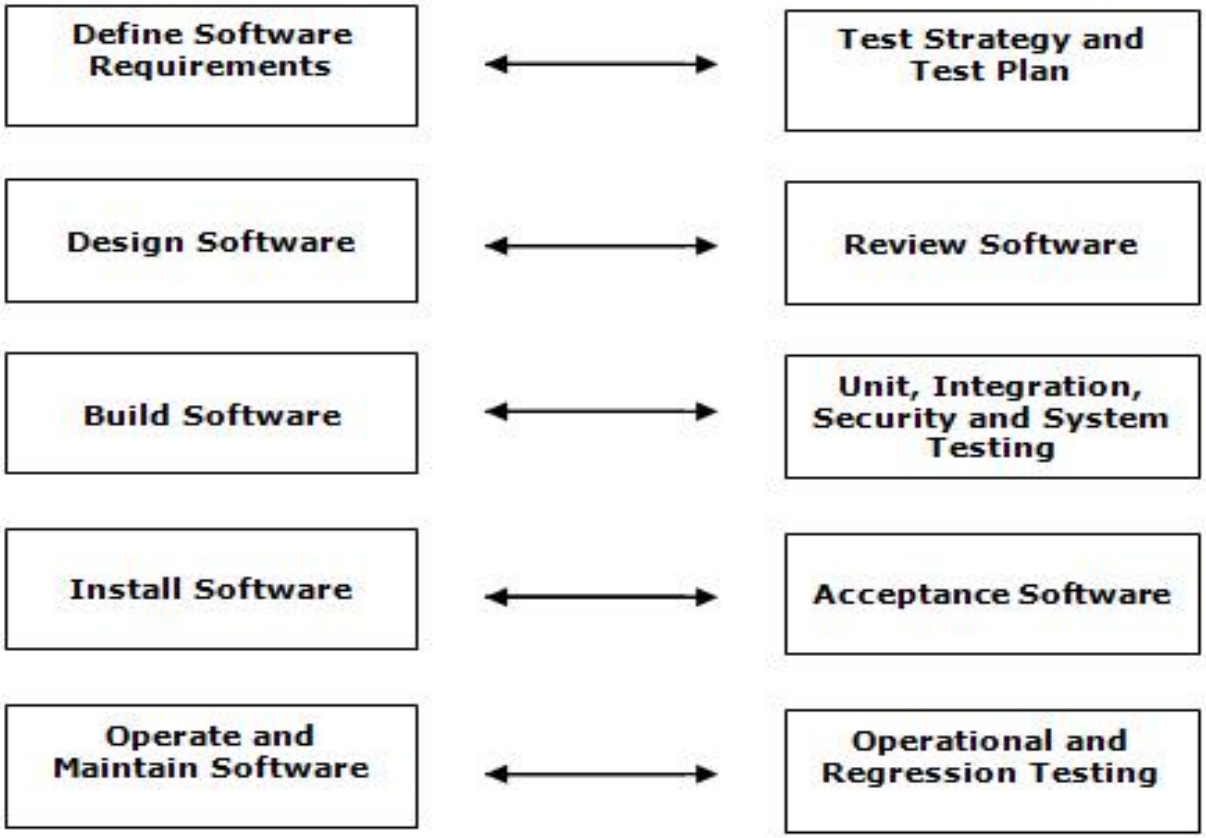
Test Plans

Covered in chapter 3

Outline

- Definition
- Test Strategy
- Test Plans
- **Software Review**
- Unit Testing
- Integration Testing
- System Testing
- Security testing
- Acceptance Test
- Regression Testing

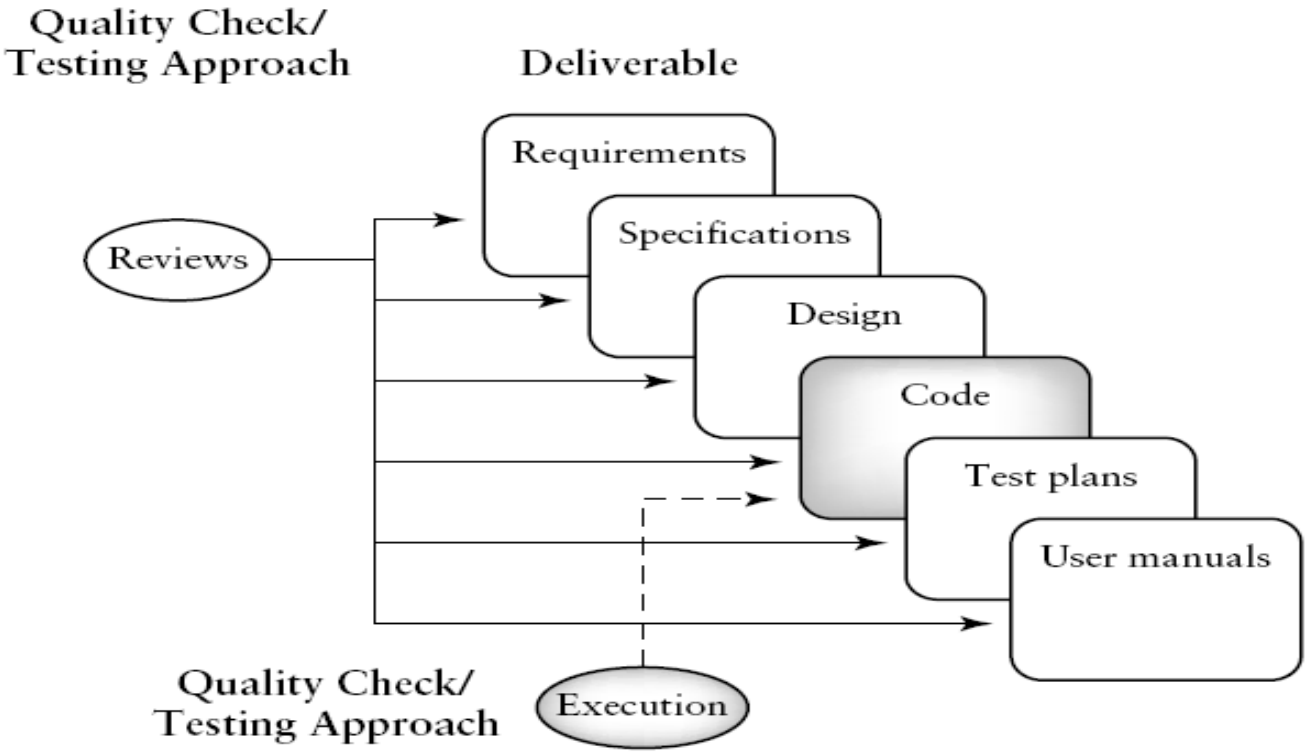
Software testing life cycle



Software Review

- Dynamic execution can only be applied to the software code. We use dynamic execution as a tool to detect defects and to evaluate quality attributes of the code.
- This testing option is not applicable for the majority of the other software artifacts. Among the questions that arise are: How can we evaluate or analyze a requirements document, a design document, a test plan, or a user manual.

Software Review



Software Review

One powerful tool that we can use is a manual static testing technique that is generally known as the technical review.

A review is a group meeting whose purpose is to evaluate a software artifact or a set of software artifacts.

Software Review

Software Review Techniques:

- Formal design reviews
- Peer reviews
- Expert opinions

Software Review

Formal design reviews

Committees whose members examine the documents presented by the development teams usually carry out formal design reviews.

Software Review

Peer reviews

Peer reviews are directed at reviewing short documents, chapters or parts of a report, a coded printout of a software module, and the like.

Peer reviews can take several forms and use many methods; usually, the reviewers are all peers, not superiors, who provide professional assistance to colleagues. The main objective of inspections and walkthroughs is to detect as many design and programming faults as possible.

Software Review

Expert opinions, prepared by outside experts, support quality evaluation by introducing additional capabilities to the internal review staff.

Software Review

The general goals for the reviewers are to:

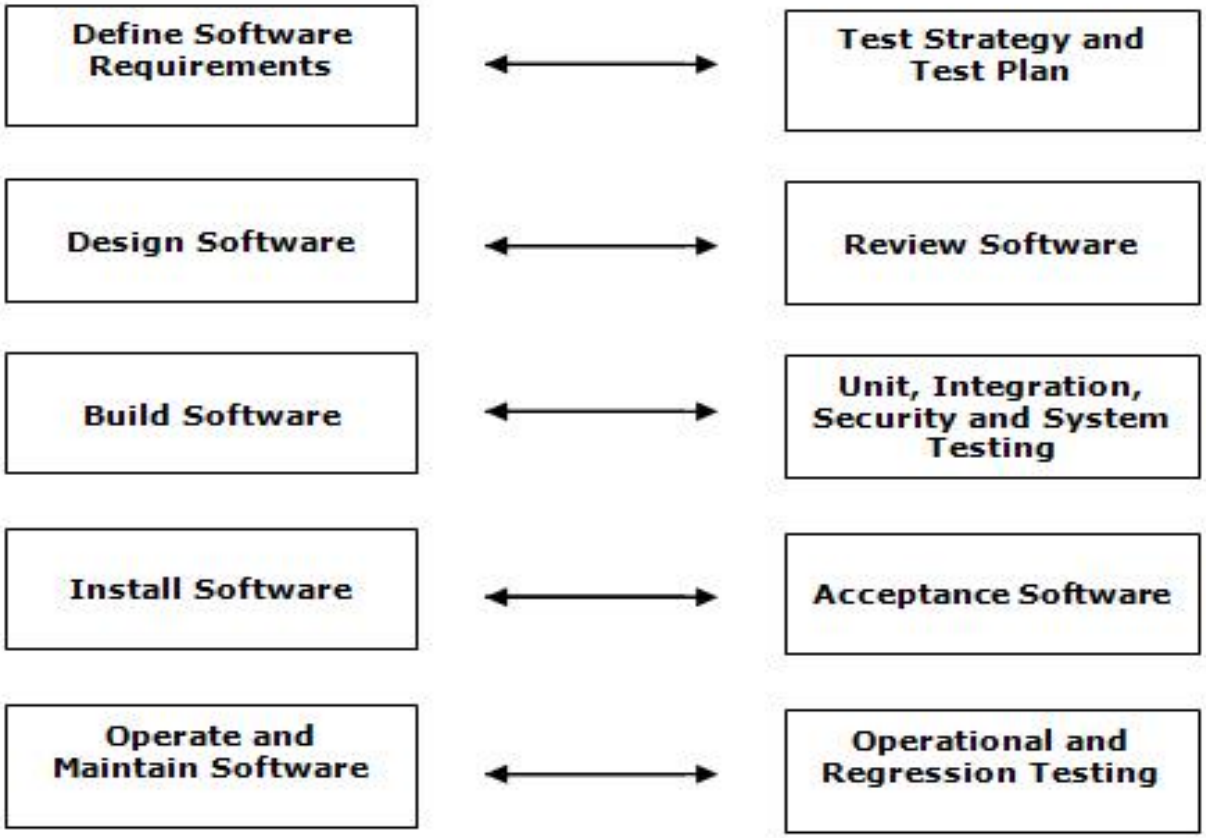
Identify problem components or components in the software artifact that need improvement;

Identify components of the software artifact that do not need improvement;

Identify specific errors or defects in the software artifact (defect detection);

Ensure that the artifact conforms to organizational standards

Software testing life cycle



Levels of testing

- Unit Testing
- Integration Testing
- System Testing

Levels of testing

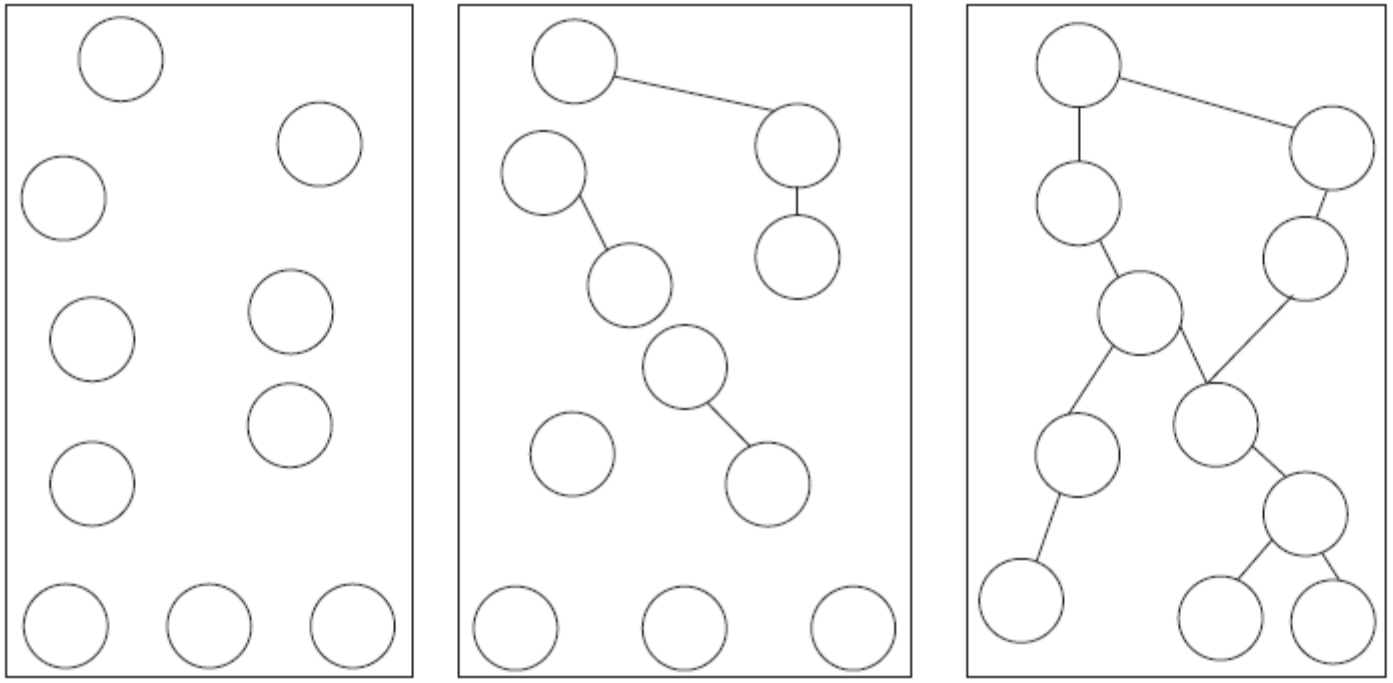


FIGURE 7.3 Levels of Testing

Outline

- Definition
- Test Strategy
- Test Plans
- Software Review
- **Unit Testing**
- Integration Testing
- System Testing
- Security testing
- Acceptance Test
- Regression Testing

Unit Testing

Unit testing is a software test method in which a programmer tests if individual units of source code are fit for use.

A unit is the smallest testable part of an application.

In procedural programming a unit may be an individual function or procedure while in object-oriented programming, the smallest unit is usually a class.

Unit Test

Unit testing is conducted in two complementary phases:

- Static unit testing
- Dynamic unit testing (Will be discussed in the next chapters)

Static Unit Test

In static unit testing (also called code review) a programmer does not execute the unit; instead, the code is examined over all possible behaviors that might arise during run time.

The code of each unit is validated against requirements of the unit by reviewing the code.

During the review process, potential issues are identified and resolved

Static Unit Test

The goal of such an exercise is to assess the quality of the software in question, *not the quality of the* process used to develop the product which done in software review.

The objective of code review is to review the code, not to evaluate the author of the code. A clash may occur between the author of the code and the reviewers, and this may make the meetings unproductive.

Static Unit Test

The composition of the review group involves a number of people with different roles. These roles are explained as follows:

Moderator: A review meeting is chaired by the moderator. The moderator is a trained individual who guides the pace of the review process. The moderator selects the reviewers and schedules the review meetings.

Static Unit Test

Author: This is the person who has written the code to be reviewed.

Presenter: A presenter is someone other than the author of the code. The presenter reads the code beforehand to understand it.

Recordkeeper: The recordkeeper documents the problems found during the review process and the follow-up actions suggested. The person should be different than the author and the moderator

Static Unit Test

Reviewers: These are experts in the subject area of the code under review. The group size depends on the content of the material under review. As a rule of thumb, the group size is between 3 and 7.

Observers: These are people who want to learn about the code under review. These people do not participate in the review process but are simply passive observers

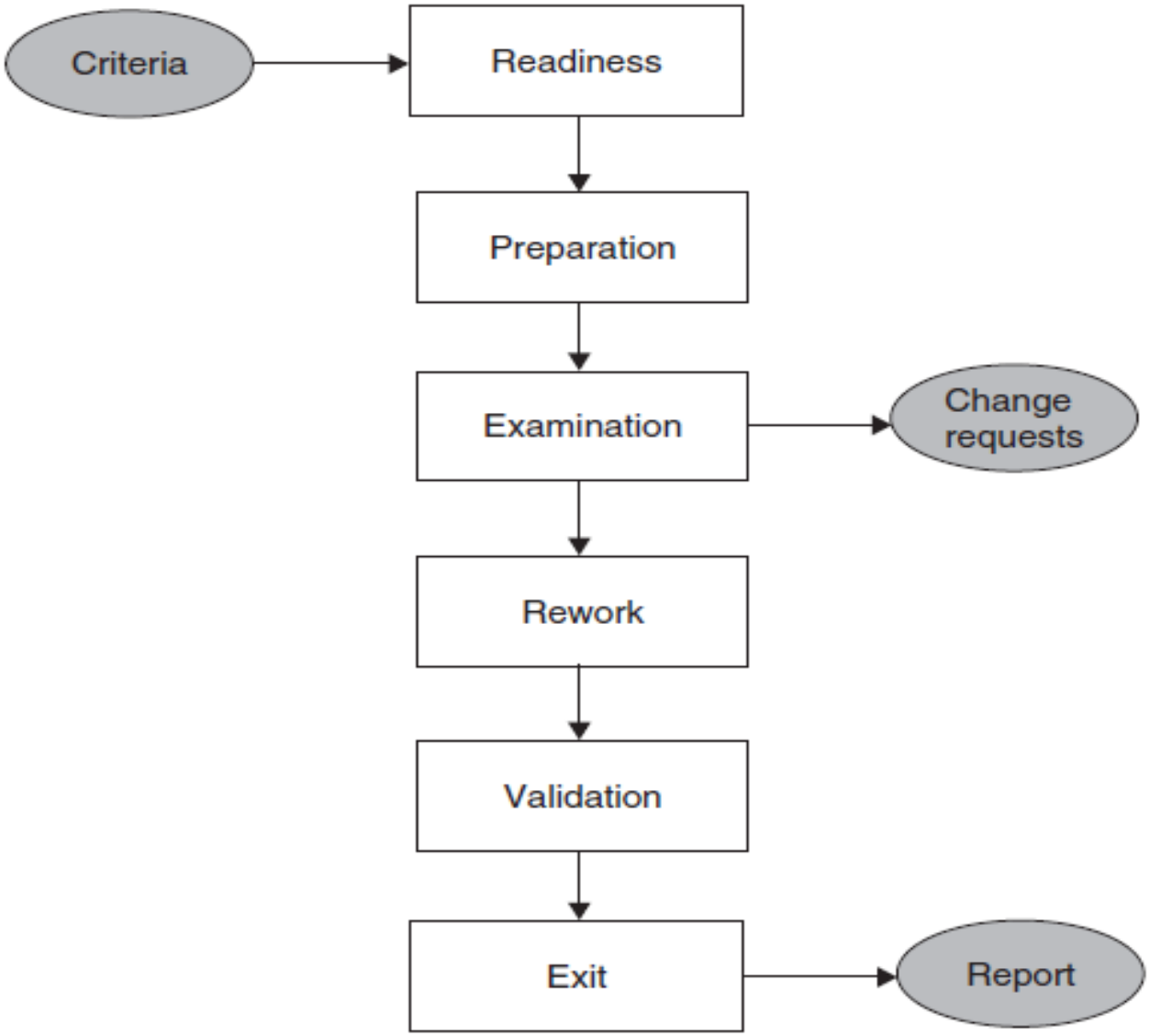


Figure: Steps in the code review process

Static Unit Test

All the people involved in the review process are informed of the group review meeting schedule two or three days before the meeting.

The general guidelines for performing code review consists of six steps as follows:

Static Unit Test

Step 1: Readiness:

The author of the unit ensures that the unit under test is ready for review.

Step 2: Preparation

Before the meeting, each reviewer carefully reviews the work package. It is expected that the reviewers read the code and understand its organization and operation before the review meeting

Static Unit Test

Step 3: Examination

The examination process consists of the following activities:

- The author makes a presentation.
- The presenter reads the code line by line.
- The recordkeeper documents the change requests and the suggestions for fixing the problems, if there are any.
- The moderator ensures that the meeting remains focused on the review process

Static Unit Test

At the end of the meeting, a decision is taken regarding whether or not to call another meeting to further review the code. If the review process leads to extensive rework of the code or critical issues are identified in the process, then another meeting is generally convened. Otherwise, a second meeting is not scheduled, and the author is given the responsibility of fixing the CRs.

Static Unit Test

Step 4: Rework

At the end of the meeting, the recordkeeper produces a summary of the meeting that includes the following information:

- A list of all the CRs, the dates by which those will be fixed, and the names of the persons responsible for validating the CRs
- A list of improvement opportunities

Static Unit Test

After the meeting, the author works on the CRs to fix the problems. The author documents the improvements made to the code in the CRs. The author makes an attempt to address the issues within the agreed-upon time frame

Static Unit Test

Step 5: Validation

The CRs are independently validated by the moderator or another person designated for this purpose.

The validation process involves checking the modified code as documented in the CRs and ensuring that the suggested improvements have been implemented correctly. The revised and final version of the outcome of the review meeting is distributed to all the group members.

Static Unit Test

Step 6: Exit

Summarizing the review process, it is said to be complete if all of the following actions have been taken:

- Every line of code in the unit has been inspected.
- If too many defects are found in a module, the module is once again reviewed after corrections are applied by the author. As a rule of thumb, if more than 5% of the total lines of code are thought to be debatable, then a second review is scheduled.

Static Unit Test

- The author and the reviewers reach a agreement that when corrections have been applied the code will be potentially free of defects.
- All the CRs are documented and validated by the moderator or someone else. The author's follow-up actions are documented.
- A summary report of the meeting including the CRs is distributed to all the members of the review group.

Unit testing

Objectives	• To test the function of a program or unit of code such as a program or module • To test internal logic • To verify internal design • To test path & conditions coverage • To test exception conditions & error handling
When	• After modules are coded
Input	• Internal Application Design • Master Test Plan • Unit Test Plan
Output	• Unit Test Report

Who	• Developer
Methods	White Box testing techniques
Tools	• Debug • Re-structure • Code Analyzers • Path/statement coverage tools
Education	• Testing Methodology • Effective use of tools

Outline

- Definition
- Test Strategy
- Test Plans
- Software Review
- Unit Testing
- Integration Testing
- System Testing
- Security testing
- Acceptance Test
- Regression Testing

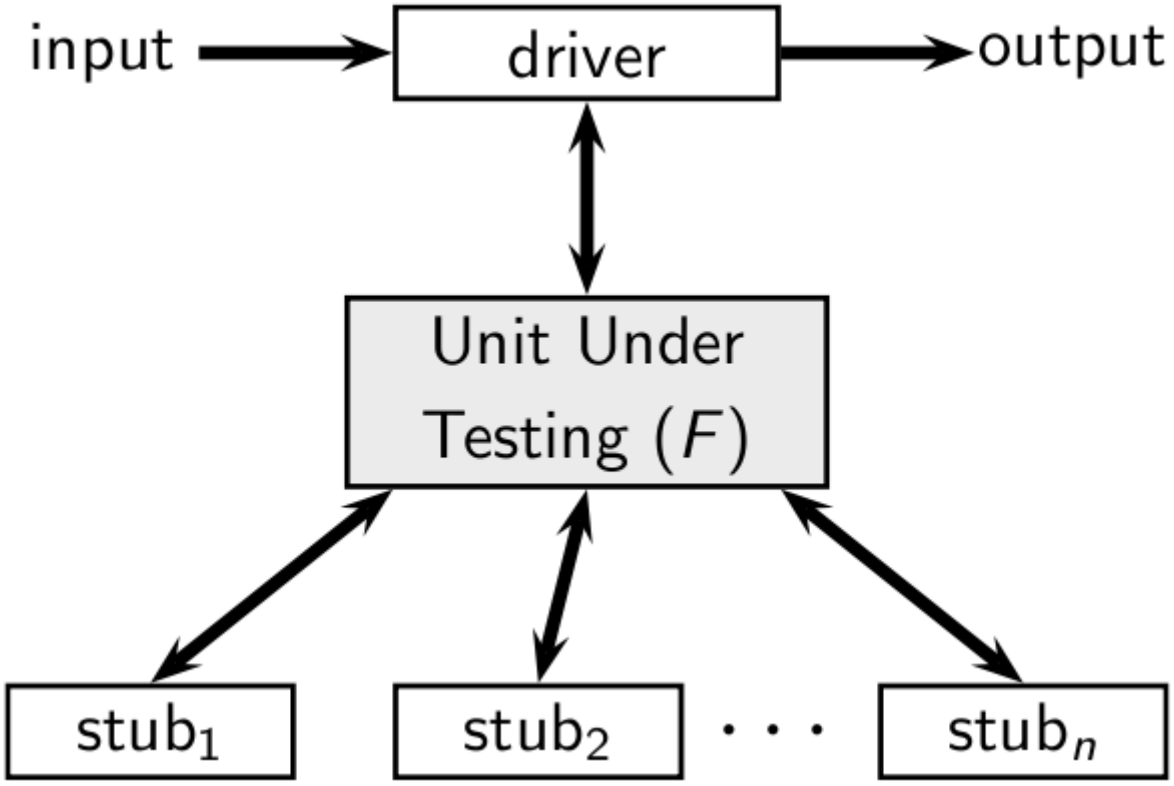
Integration testing

- **Integration testing** is the activity of software testing in which individual software modules are combined and tested as a group.
- It occurs after unit testing and before system testing.
- Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Drivers and Stubs

Stubs and drivers are dummy programs written while integration testing.

Drivers and Stubs

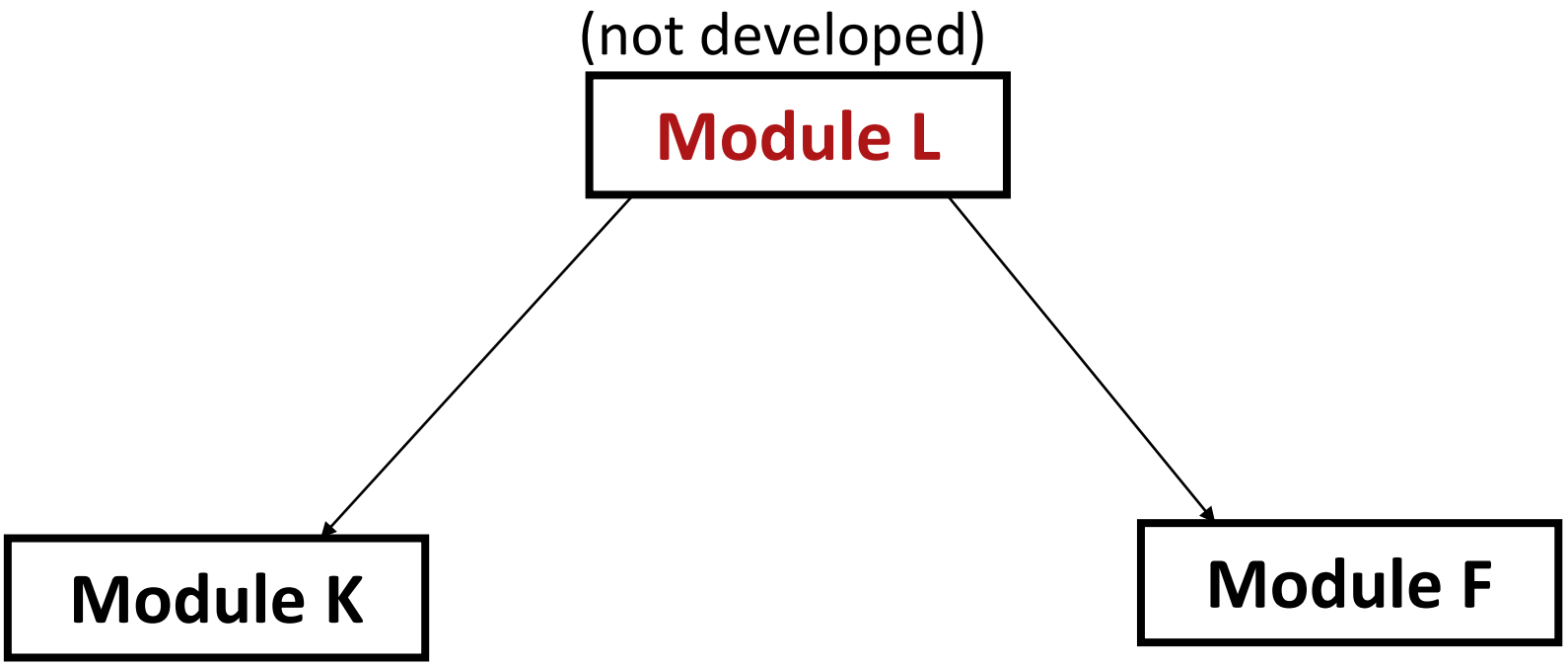


Drivers and Stubs

Driver: A program that calls the interface procedures of the module being tested and reports the results

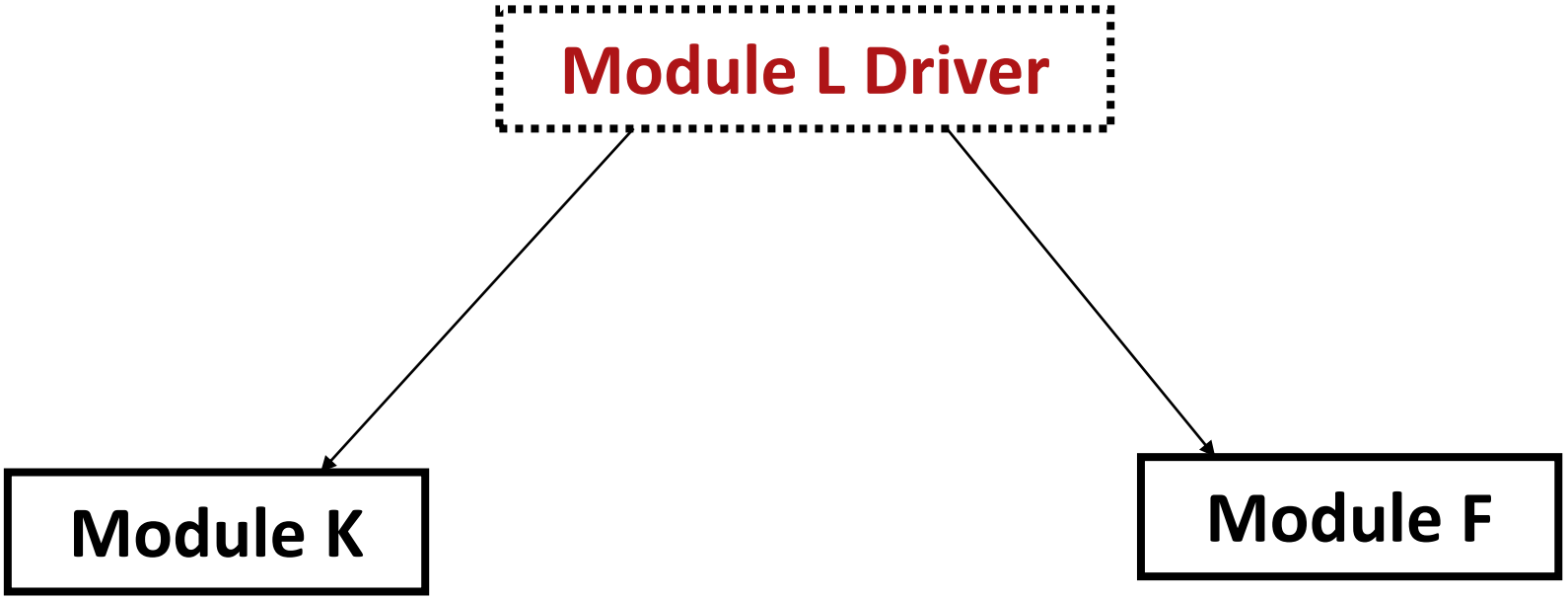
- A driver simulates a module that calls the module currently being tested.

Drivers and Stubs



Drivers and Stubs

Dummy code that **returns** values from Module L and Module K



Drivers and Stubs

```
void functionThatCallsPrice (params..) { //this is the driver
    int p = price(param1);
    printf("Price is: %d", p);
}
void price(int param) {
    //complex ecuations and DB interogations that determin the real price
}
```

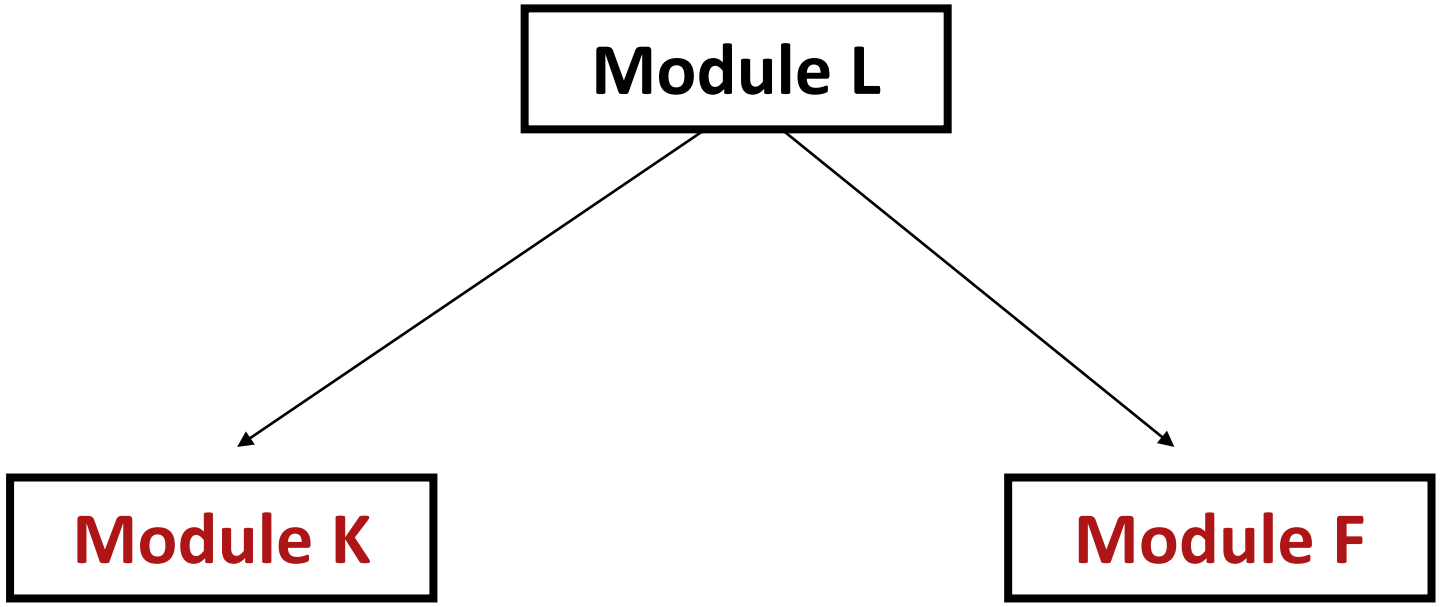
Drivers and Stubs

Stub: A program that has the same interface procedures as a module that is being called by the module being tested but is simpler.

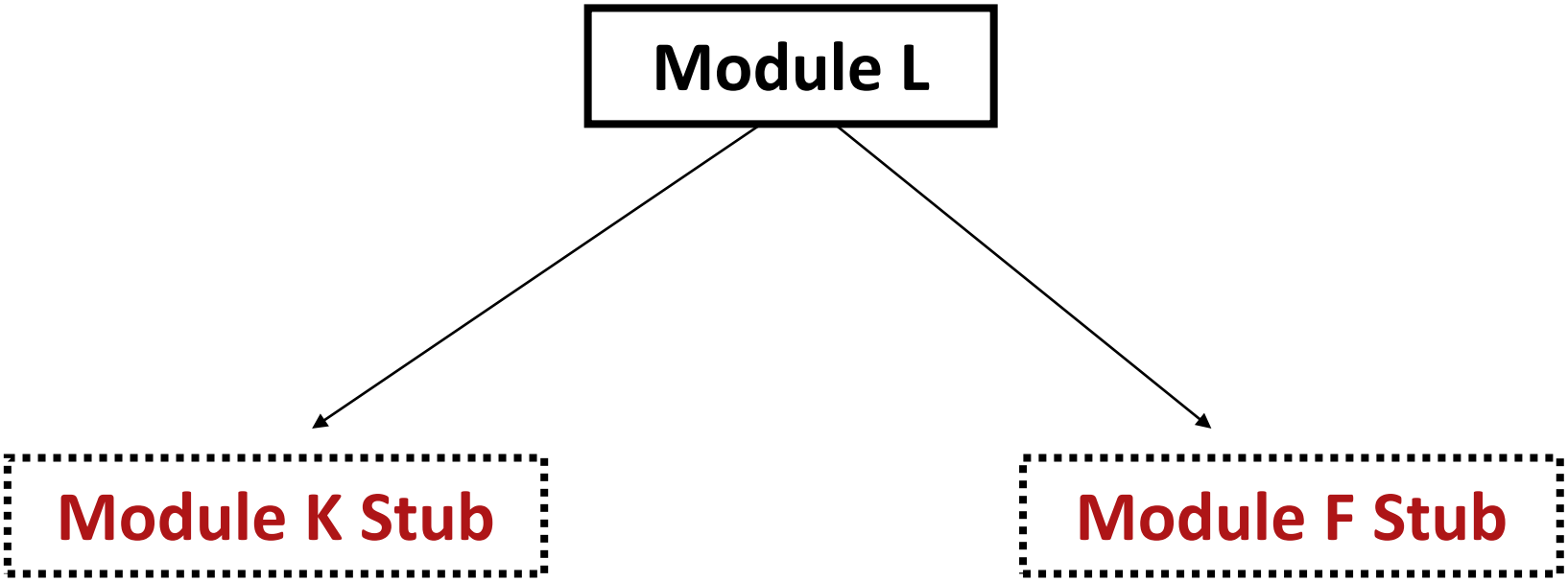
- A stub simulates a module called by the module currently being tested

Drivers and Stubs

So how can we test **MODULE L?**



Drivers and Stubs



Dummy code that **SIMULATES** the functionality of the undeveloped modules

Drivers and Stubs

```
void functionWeTest(params..) {

    ……..

    int p = price(param1);

        ……..

}

void price(int param) { //this is the stub

    return 10; // We don't care what the price is. We just need a value so we can test the other function

}
```

Bottom-Up Integration

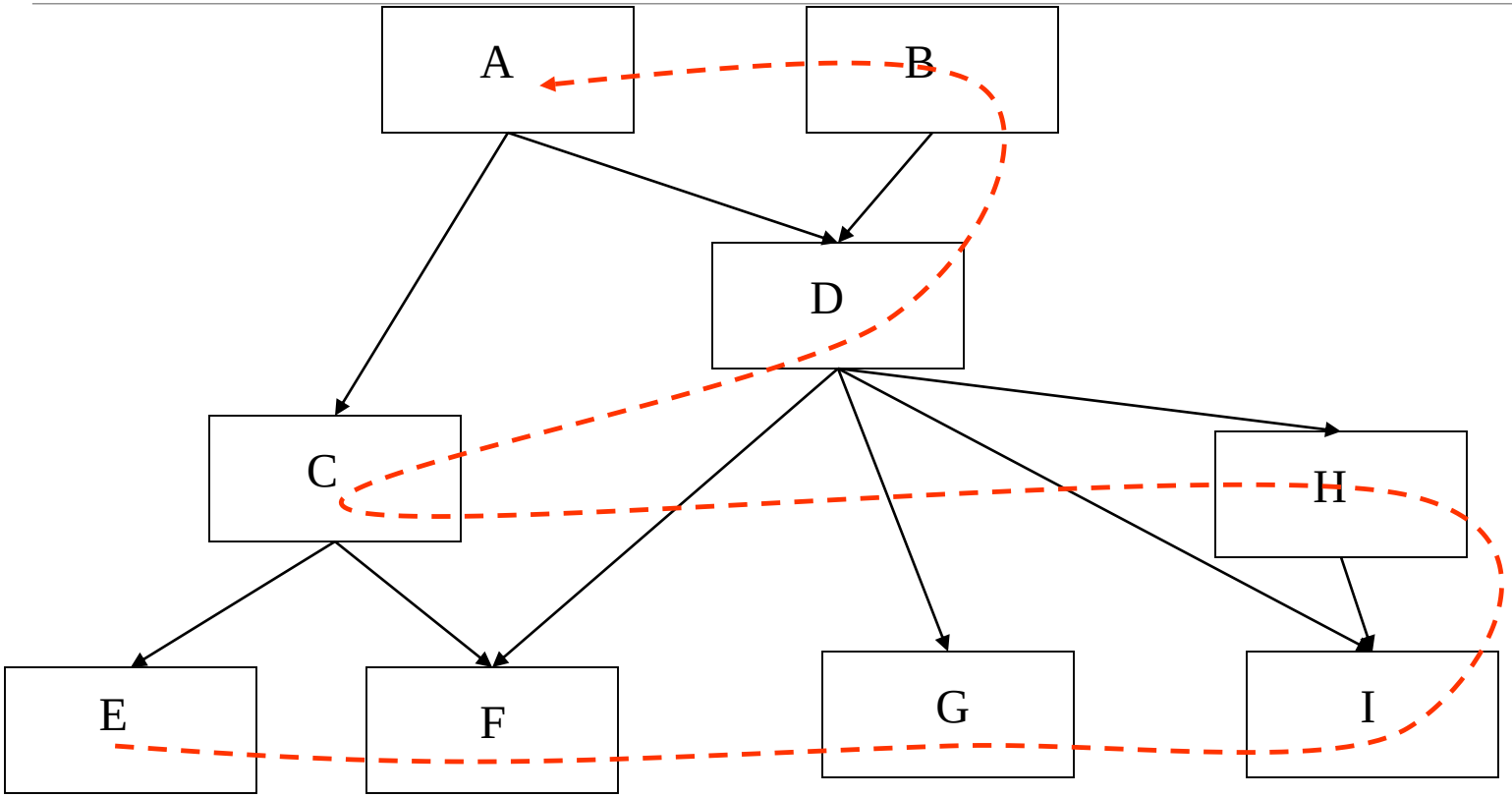
Integrate individual components in levels until the complete system is created

Only terminal modules are tested in isolation

Modules at lower levels are tested using the previously tested higher level modules

- This is done repeatedly until all subsystems are included in the testing

Bottom-up Integration

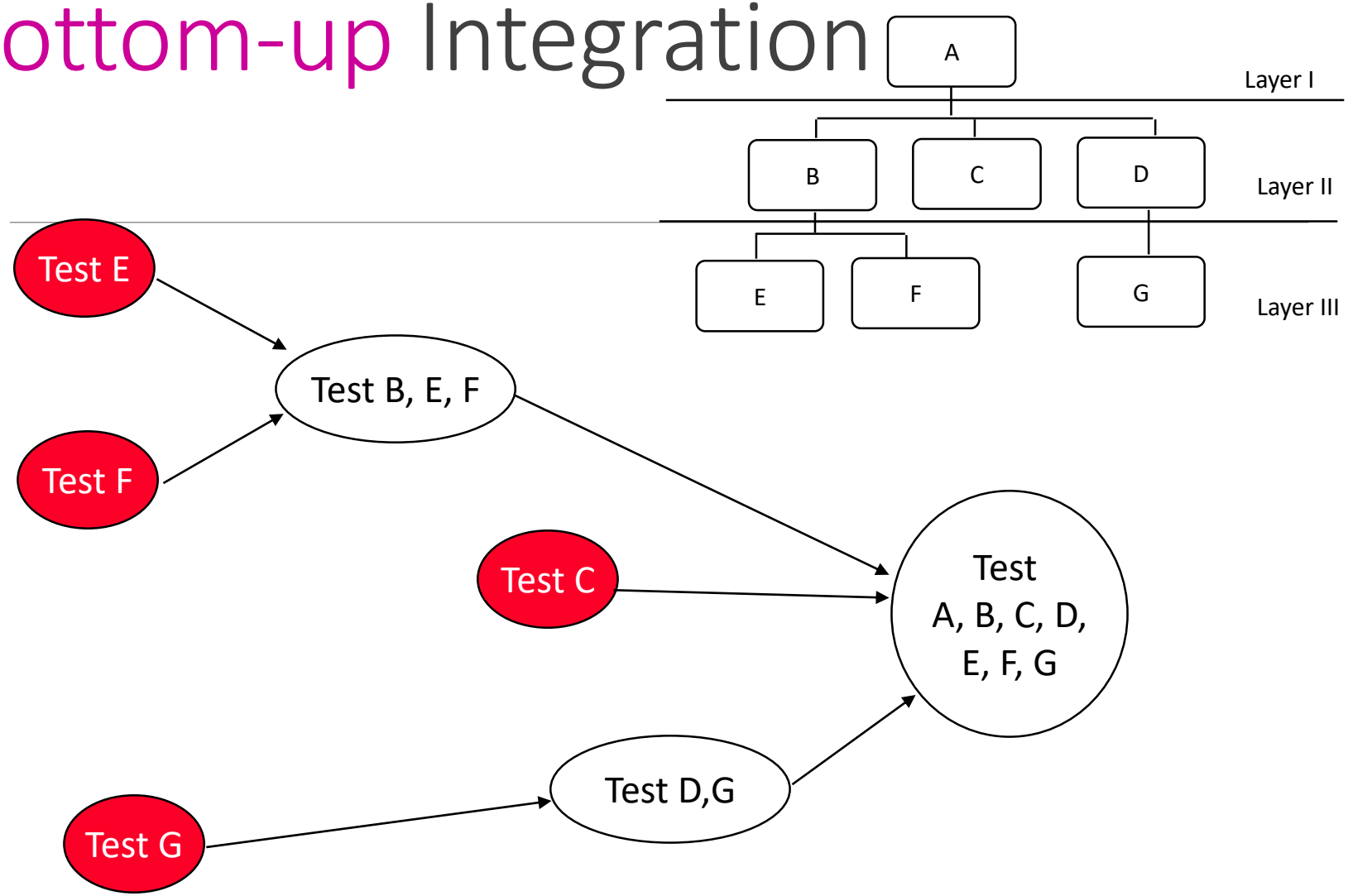


Bottom-Up Integration

Requires a module driver for each module to feed the test case input to the interface of the module being tested

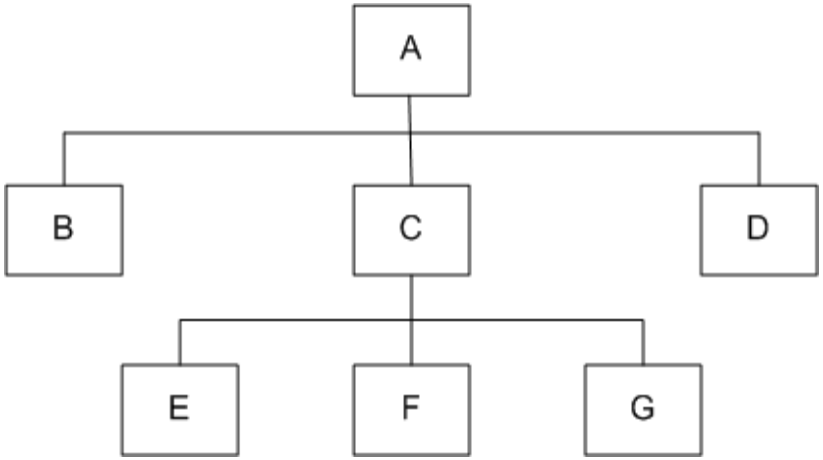
However, stubs are not needed since we are starting with the terminal modules and use already tested modules when testing modules in the lower levels

Bottom-up Integration



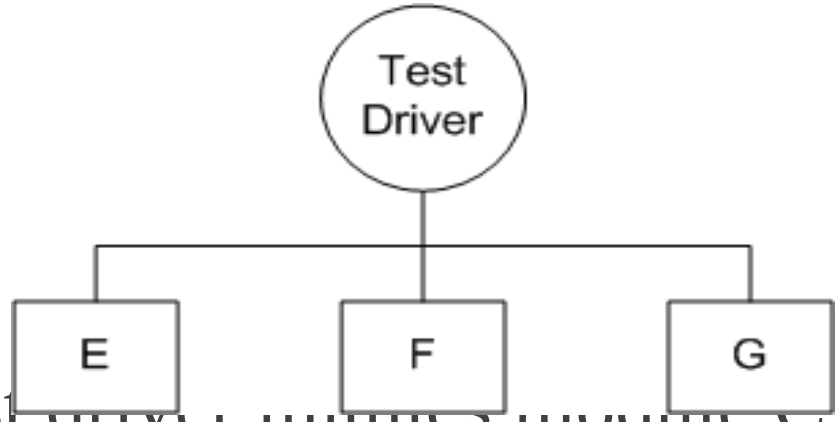
Bottom-Up Integration

Given the following system:



Bottom-Up Integration

We design a test driver to integrate lowest-level modules E, F, and G



The test driver mimics module C to integrate E, F, and G in a limited way.

Bottom-Up Integration

The test driver is replaced with actual module C.

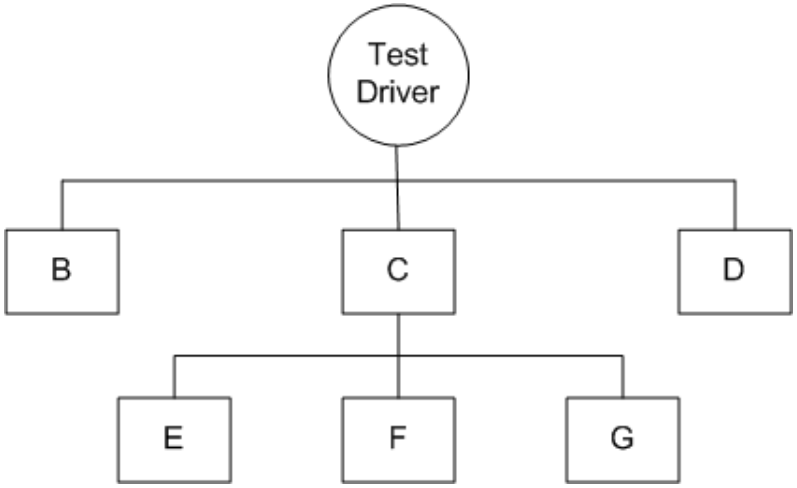
A new test driver is used

At this moment, more modules such as B and D are integrated

A new test driver is used

At this moment, more modules such as B and D are integrated

Bottom-Up Integration



The new test driver mimics the behavior of module A

Finally, the test driver is replaced with module A and further test are performed

Bottom-Up Integration

Bottom-up advantages

- Separately debugged modules
- System test by integrating previously debugged modules
- Testing upper-level modules is easier

Bottom-up disadvantages

- Drivers must be written
- Upper-level, critical modules are built last
- Drivers are more difficult to write than stubs

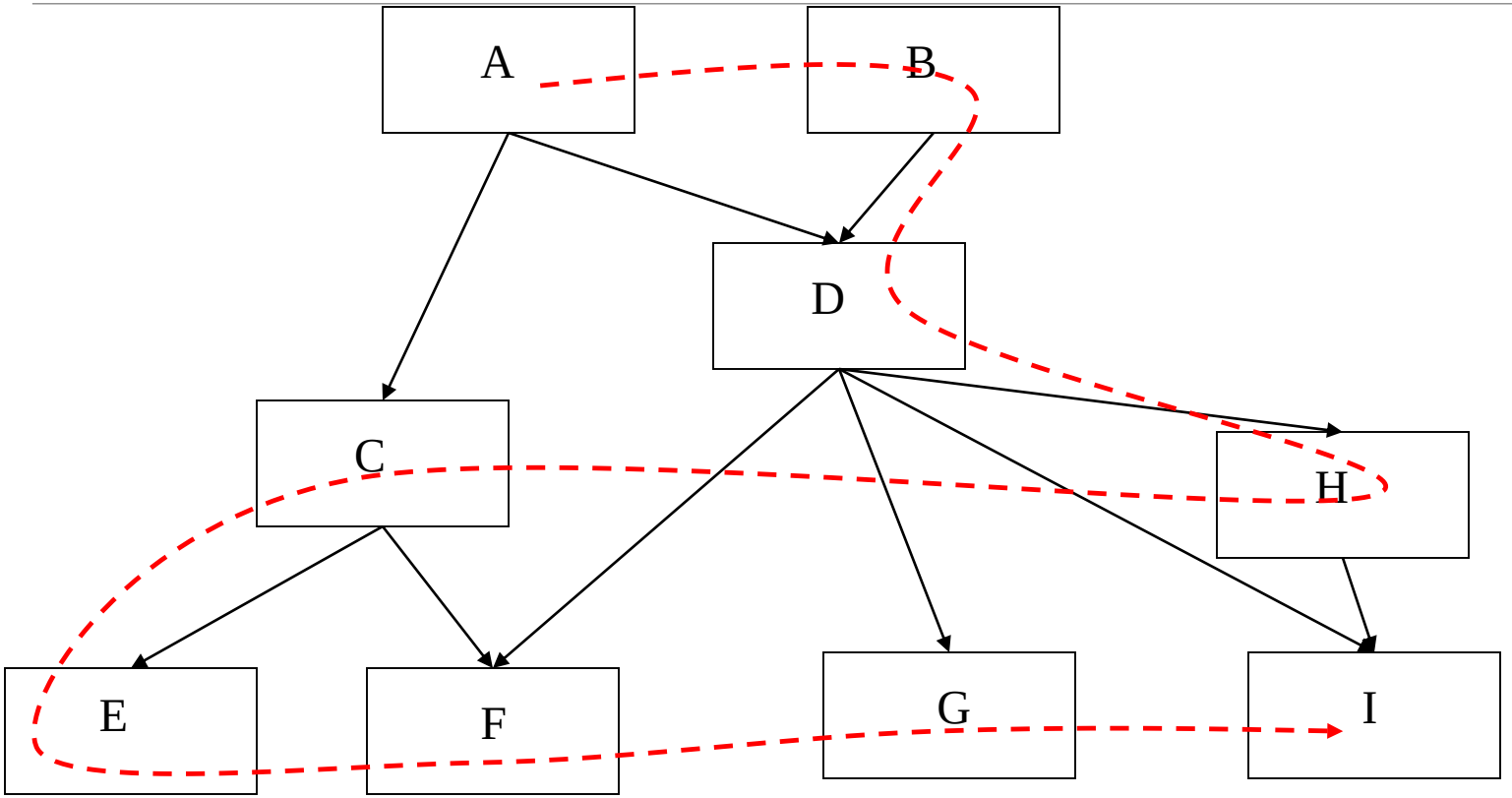
Top-down Integration

- Start with high-level system and integrate from the top-down, replacing individual components by stubs where appropriate.
- Only modules tested in isolation are the modules which are at the highest level

After a module is tested, the modules directly called by that module are merged with the already tested module and the combination is tested

- Do this until all subsystems are incorporated into the test

Top-down Integration

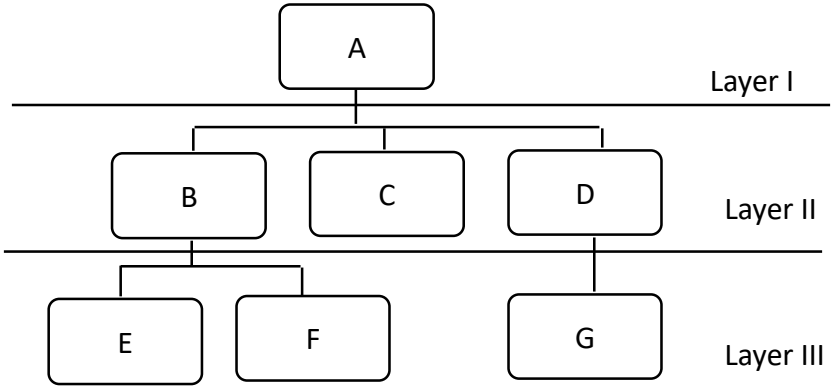


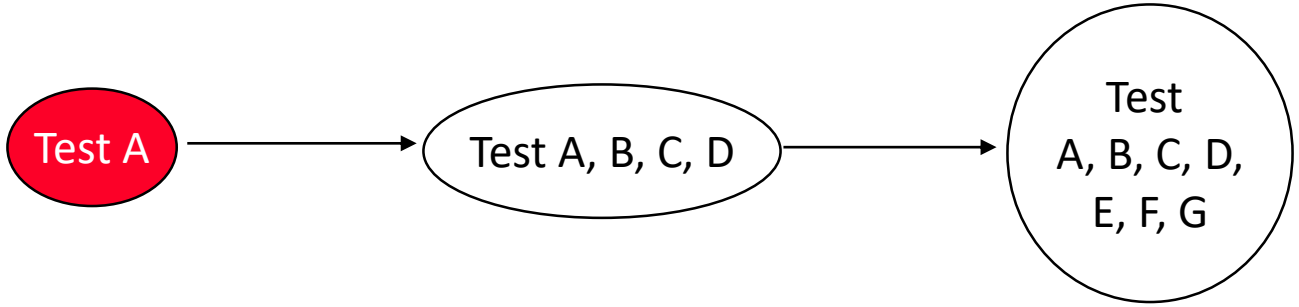
Top-down Integration

Requires stub modules to simulate the functions of the missing modules that may be called

- However, drivers are not needed since we are starting with the modules which is not used by any other module and use already tested modules when testing modules in the higher levels

Top-down Integration Testing





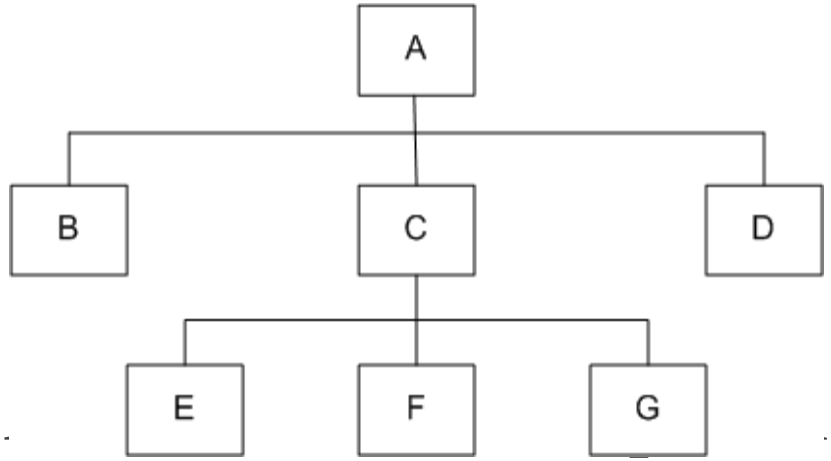
Layer I

Layer I + II

All Layers

Top-down Integration

Given the following system:

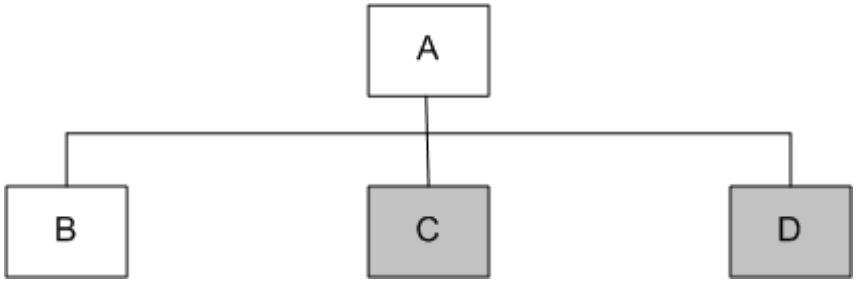


Module . . into modules B, C, and D

Modules B, D, E, F, and G are terminal modules

Top-down Integration

First integrate modules A and B using stubs C` and D` (represented by grey boxes)

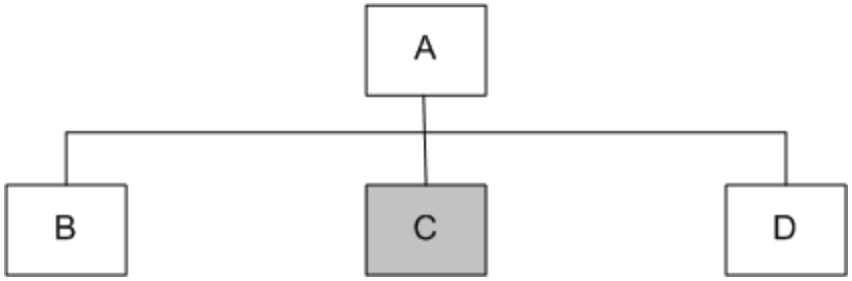


Top-down Integration

Next stub D` has been replaced with its actual instance D

Two kinds of tests are performed:

- Test the interface between A and D
- Regression tests to look for interface defects between A and B in the presence of module D

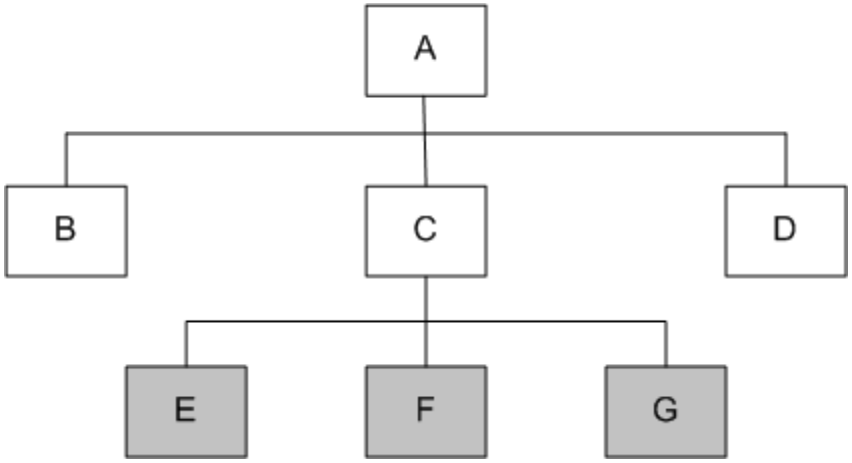


Top-down Integration

Stub C` has been replaced with the actual module C, and new stubs E`, F`, and G`

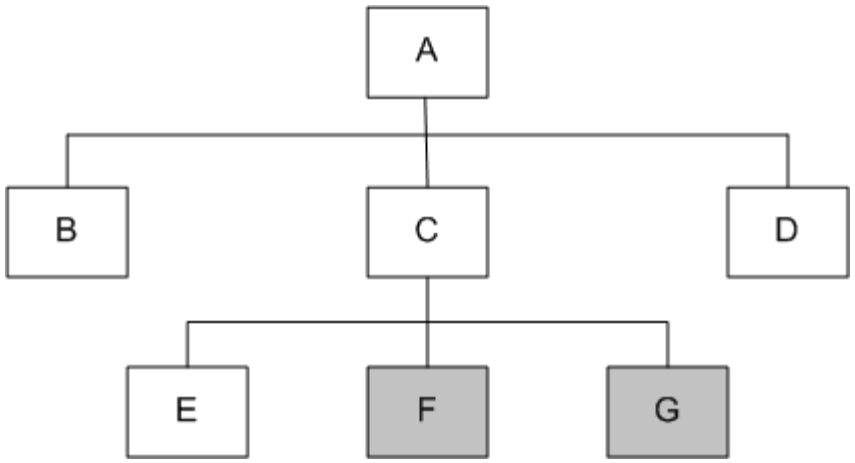
Perform tests as follows:

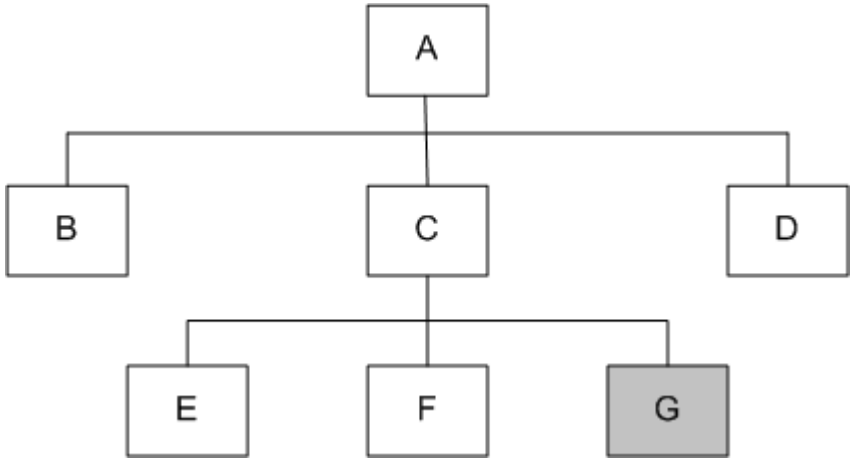
- first, test the interface between A and C;
- second, test the combined modules A, B, and D in the presence of C



Top-down Integration

The rest of the process depicted as follows





Top-down Integration

Top-down advantages

- Separately debugged modules
- System test by integrating previously debugged modules
- Stubs are easier to code than drivers

Top-down disadvantages

- Stubs must be written
- Low-level, critical modules built last
- Testing upper-level modules is difficult

Sandwich Testing Strategy

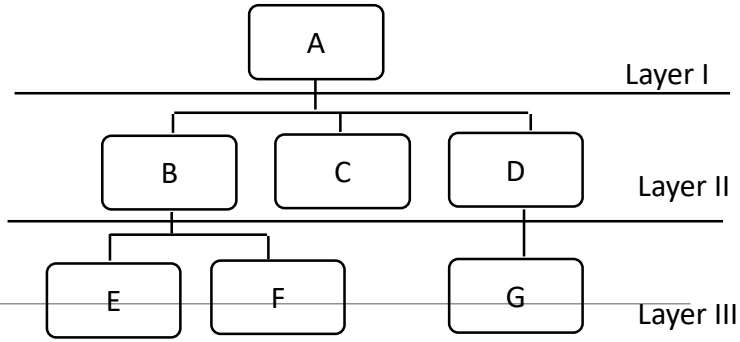
Combines top-down strategy with bottom-up strategy

The system is view as having three layers

- A target layer in the middle
- A layer above the target
- A layer below the target
- Testing converges at the target layer

Top and Bottom Layer Tests can be done in parallel

Sandwich Testing Strategy



Test E

Test F

Test G

Bottom Layer Tests

Test B, E, F

Test D,G

Bottom up testing

Test A, B, C, D, E, F, G

Test A,B,C, D

Test A

Top Layer Tests

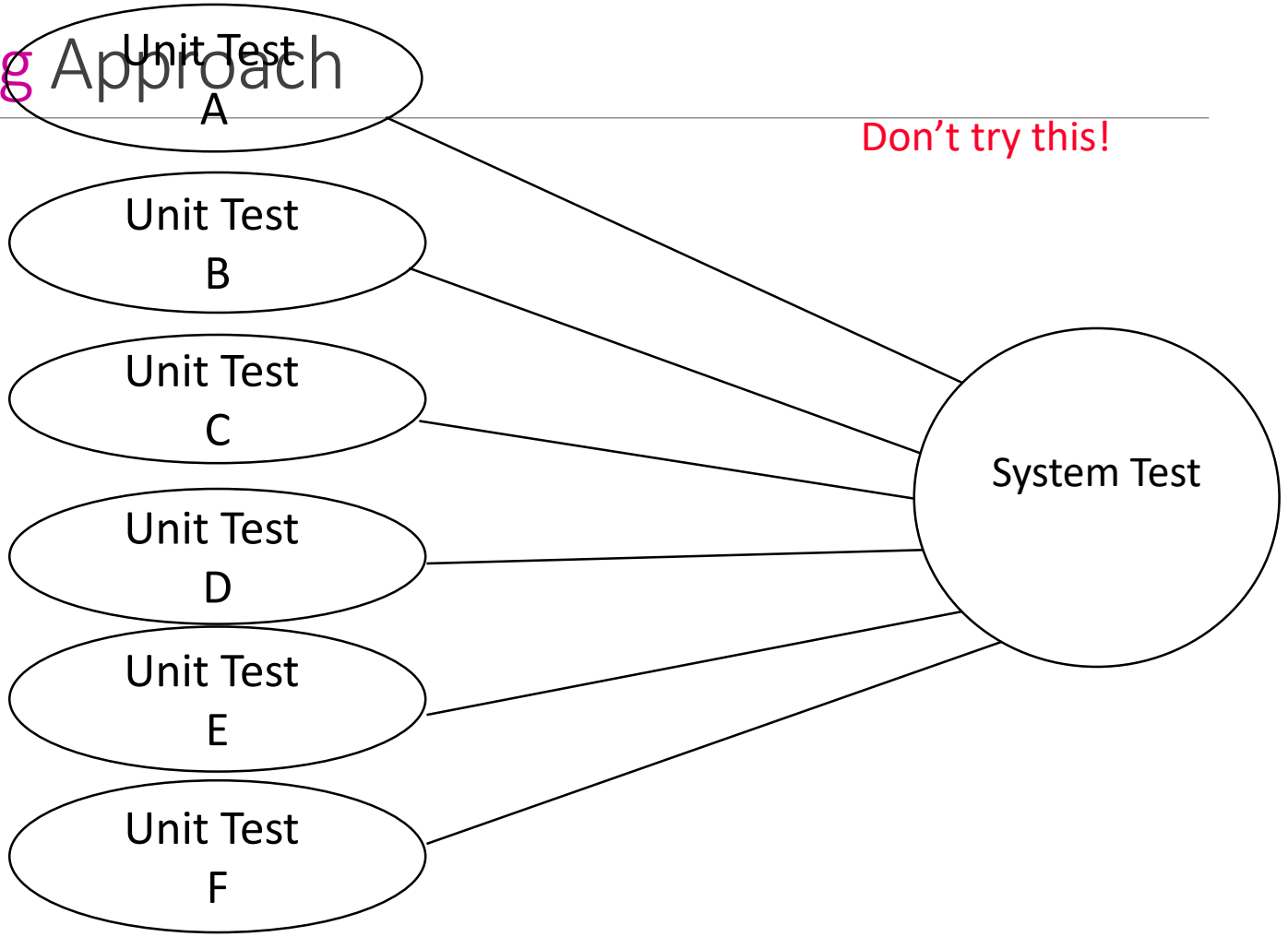
Top down

Big Bang Integration

Big Bang Integration

- Every module is unit tested in isolation
- After all of the modules are tested they are all integrated together at once and tested

Big-Bang Approach



Integration testing

Objectives	• To technically verify proper interfacing between modules, and within sub-systems
When	• After modules are unit tested
Input	• Internal & External Application Design • Master Test Plan • Integration Test Plan
Output	• Integration Test report

Who	• Developers
Methods	• White and Black Box techniques • Problem / Configuration Management
Tools	• Debug • Re-structure • Code Analyzers
Education	• Testing Methodology • Effective use of tools

Outline

- Definition
- Test Strategy
- Test Plans
- Software Review
- Unit Testing
- Integration Testing
- System Testing
- Security testing
- Acceptance Test
- Regression Testing

System testing

System testing of software is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

System testing

Function testing: does the integrated system perform as promised by the requirements specification? (Will be discussed in)

Some types of System Testing which is no-chapter 6functional:

- Usability Testing (will be discussed in quality factors)
- Compatibility testing
- Load Test
- Stress Test

Compatibility testing

Compatibility testing: is conducted on the application to evaluate the application's compatibility with the computing environment such as Computing capacity of Hardware Platform, Bandwidth handling capacity of networking hardware, Compatibility of peripherals ,Operating systems.

Testing how well software performs in a particular HW/SW/OS/NW environment.

Compatibility Testing

For Example in web site compatibility testing includes:

Hardware platform
- Mac, PC, PDA, WiFi wristwatch?

Browser software version
- Firefox , IE , Pocket IE, Netscape , Safari

Browser plug-ins
- To play specific types of audio or video files

Browser options
- Security options, ALT text, plug-in, pop ups

Video resolution and color depth
- 640x480, 800x600, 1024x768, 1280x1024, 256 colors, 16 colors

Text size
- Small fonts, medium fonts, large fonts

Connection speed
- DSL, modems of varying speed.

Load Testing

A load test is usually conducted to understand the behavior of the system under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration.

Load Testing

This test will give out the response times of all the important business critical transactions. If the database, application server, etc. are also monitored, then this simple test can itself point towards any bottlenecks in the application software.

Stress testing

Stress testing is normally used to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum

Difference between Load and Stress Testing

Load Testing

- Process of exercising the system under test by feeding it the largest tasks it can operate with.
- Constantly increasing the load on the system via automated tools to simulate real time scenario with virtual users.

Examples:

- Testing a word processor by editing a very large document.
- For Web Application load is defined in terms of concurrent users or HTTP connections.

Stress Testing

Trying to break the system under test by overwhelming its resources or by taking resources away from it.

Purpose is to make sure that the system fails and recovers gracefully.

Example:

Double the baseline number for concurrent users/HTTP connections.

Randomly shut down and restart ports on the network switches/routers that connects servers.

System Testing

Objectives	• To verify that the system components perform control functions • To perform inter-system test • To demonstrate that the system performs both functionally and operationally as specified.
When	• After Integration Testing
Input	• Detailed Requirements & External Application Design • Master Test Plan • System Test Plan
Output	• System Test Report

Who	• Testers and Users
Methods	• Black Box • Problem / Configuration Management
Tools	• Recommended set of tools
Education	• Testing Methodology • Effective use of tools

Security testing

Security testing Protect Data and Systems from Malicious entities

Security testing is carried out when some important information and assets managed by the software application are of significant importance to the organization.

Failures in the software security system can be serious especially when not detected, thereby resulting in a loss or compromise of information without the knowledge of that loss

Security testing

Test cases aimed at uncovering security vulnerabilities

The security testing should be performed both prior to the system going into the operation and after the system is put into operation

Security testing

The objective of security testing is to demonstrate the following:

1) The software behaves securely and consistently under all conditions—both expected and unexpected.

2) If the software fails, the failure does not leave the software, its data, or its resources to attack.

3) Vague areas of code and inactive functions cannot be compromised or exploited.

Security testing

4) Interfaces and interactions among components at the application, framework/ middleware, and operating system levels are consistently secure.

5) Exception and error handling mechanisms resolve all faults and errors in ways that do not leave the software, its resources, its data, or its environment vulnerable to unauthorized modification or denial-of-service attack.

Security testing

Useful types of security tests include the following:

1) Verify that only authorized accesses to the system are permitted. This may include authentication of user ID and password and verification of expiry of a password. Other Biometrics can be used.

2) Verify the correctness of both encryption and decryption algorithms for systems where data/messages are encoded.

Security testing

3) Verify that illegal reading of files, to which the perpetrator is not authorized, is not allowed.

4) Ensure that virus checkers prevent or curtail entry of viruses into the system.

5) Ensure that the system is available to authorized users when a zero-day attack occurs.

(Zero-day attacks occur during the vulnerability window that exists in the time between when a vulnerability is first exploited and when software developers start to develop a counter to that threat.)

Security testing

6) Try to identify any “backdoors” in the system usually left open by the software developers. Buffer overflows are the most commonly found vulnerability in code that can be exploited to compromise the system. Try to break into the system by exploiting the backdoors.

Acceptance Test

Acceptance testing is a final stage of testing that is performed on a system prior to the system being delivered to a live environment

Acceptance tests represent the customer's interests. The acceptance tests give the customer confidence that the application has the required features and that they behave correctly.

Acceptance Test

In theory when all the acceptance tests pass the project is done.

Acceptance testing is a formal testing conducted by customers to determine whether a system satisfies its acceptance criteria

Acceptance testing can be conducted over a period of weeks or months, thereby uncovering errors that might degrade the system over time.

Acceptance test and Usability test

Both acceptance test and usability tests are performed by customers or end users.

Acceptance test is to validate that the requirements have been fulfilled.

Usability testing is a type of test that is performed to test how easy it is for the customer to use the end-user interface (EI).

Acceptance Test

Types of Acceptance test is:

1) User Acceptance test
   - Alpha Test
   - Beta Test

User Acceptance test

User acceptance testing (UAT), is the term used when the acceptance tests are performed by the person or persons who will be using the live system once it is delivered. If the system is being built or developed by an external supplier, this is sometimes called customer acceptance testing (CAT).

User Acceptance test

The UAT or CAT acts as a final confirmation that the system is ready for go-live. A successful acceptance test at this stage may be a contractual requirement prior to the system being signed off by the client.

Alpha Test

The alpha test is conducted at the developer's site by the customer under the project team's guidance.

In this test, users test the software on the development platform and point out errors for correction. However, the alpha test, because a few users on the development platform conduct it, has limited ability to expose errors and correct them.

Alpha Test

Alpha tests are conducted in a controlled environment. It is a simulation of real-life usage.

Once the alpha test is complete, the software product is ready for transition to the customer site for implementation and development

Beta Test

Beta testing is the system testing performed by a selected group of friendly customers.

Beta tests are conducted at the customer site in an environment where the software is exposed to a number of users.

Beta Test

The developer may or may not be present while the software is in use. So, beta testing is a real-life software experience without actual implementation.

In this test, end users record their observations, mistakes, errors, and so on and report them periodically.

Acceptance Testing

Objectives	• To verify that the system meets the user requirements
When	• After System Testing
Input	• Business Needs & Detailed Requirements • Master Test Plan • User Acceptance Test Plan
Output	• User Acceptance Test report

Who	Customer/ Bossiness / End Users
Methods	• Black Box techniques • Problem / Configuration Management
Tools	Compare, keystroke capture & playback
Education	• Testing Methodology • Effective use of tools • Product knowledge • Business Release Strategy

Operational Test

Testing conducted to evaluate a system or component in its operational environment or an exact copy of the environment if the environment is in use with the live system.

For any business critical system, operational testing should always take place. However, this type of testing is often difficult to fit into the development and testing lifecycle, as operational environment not available until very near the live date

Operational Test

Objectives of Operational Test

1) Determine that System can execute normally during operating the system
2) Determine completeness of documentation of computer operator.
3) Evaluate that operator training is complete.
4) Ensure that support mechanism has been prepared and working properly.
5) Check that operators can operate the computer system properly.

Regression Testing

Regression testing refers to the portion of the test cycle in which a program is tested to ensure that changes do not affect features that are not supposed to be affected.

Regression testing usually refers to testing activities during software maintenance phase.

Regression Testing

In this category, new tests are not designed. Instead, test cases are selected from the existing pool and executed to ensure that nothing is broken in the new version of the software

Regression testing

Version 1	Version 2
1. Develop P	1. Modify P to P′
2. Test P	2. Test P′ for new functionality
3. Release P	3. Perform regression testing on P′ to ensure that the code carried over from P behaves correctly
	4. Release P′

Regression Testing

Regression testing at different levels:

- Regression testing at the unit level
- Re-integration
- Regression testing at the system level