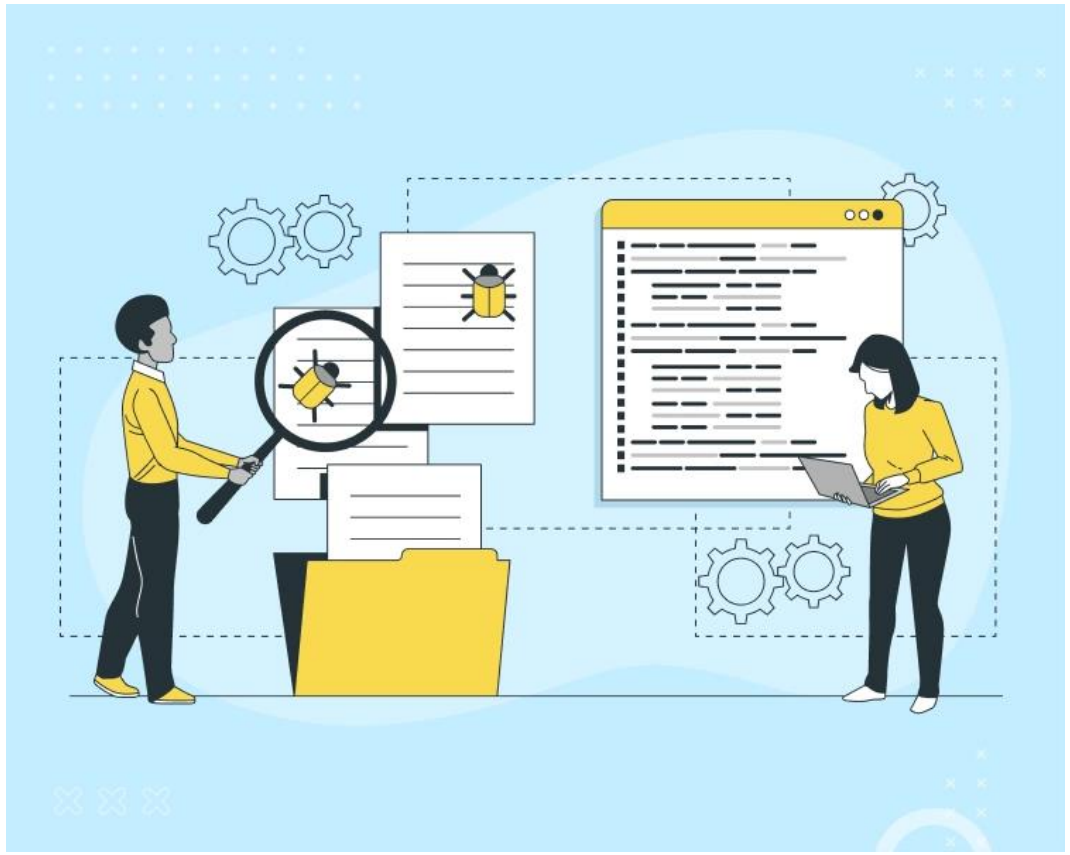


Types of Testing

PMIT 6111: ST &QA


Adding new feature to the
existing code

What Testing Shows

Errors/bugs/defects

Requirements conformance

Performance

An indication of quality

Program testing goals

- To demonstrate to the developer and the customer that the software meets its requirements.
  - For custom software, this means that there should be at least one test for every requirement in the requirements document.
  - For generic software products, it means that there should be tests for all of the system features, plus combinations of these features, that will be incorporated in the product release.
- To discover error or the situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
  - Defect testing is concerned with rooting out undesirable system behavior such as
    - system crashes
    - unwanted interactions with other systems
    - incorrect computations
    - data corruption.

Principles of Testing

1. Testing Shows the Presence of Defects, Not Their Absence
2. Exhaustive Testing is Impossible
3. Early Testing Saves Time and Money
4. Defects Cluster Together -- approximately 80% of the problems are found in 20% of the modules.
5. Beware of Pesticide Paradox
6. Testing is Context-dependent
7. Absence-of-errors is a Fallacy

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

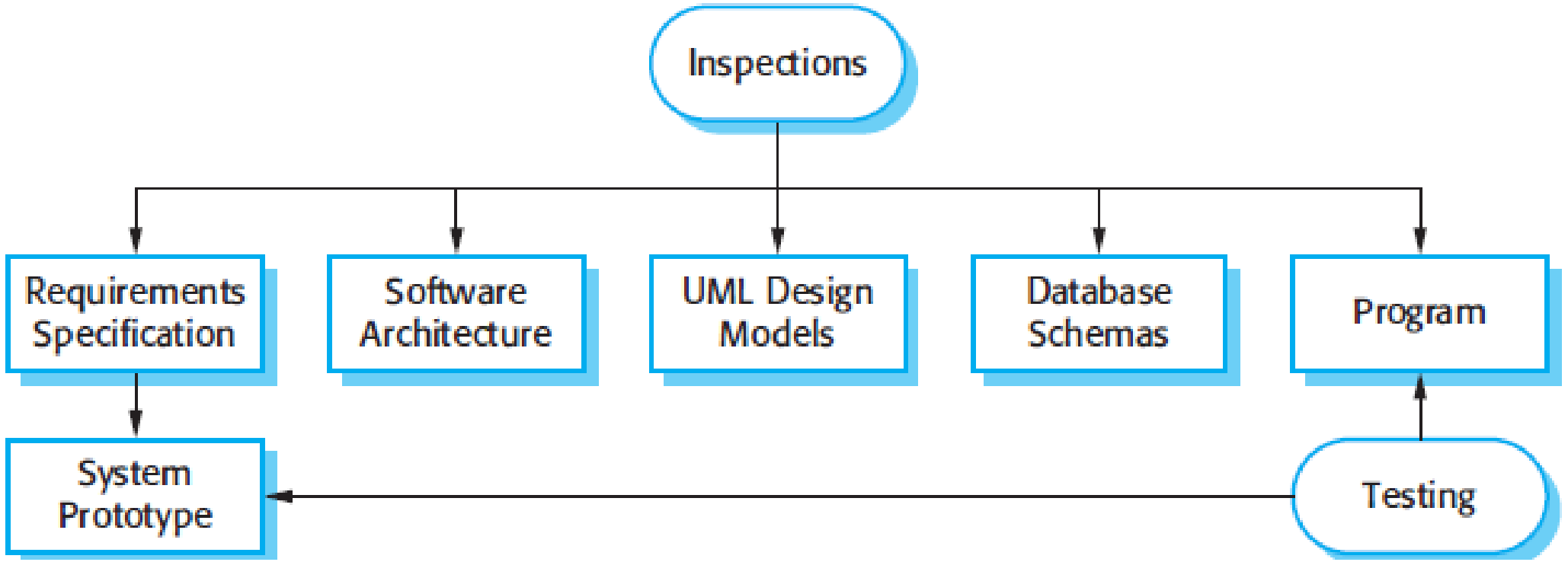
and, is driven by quality

Inspections and testing

- Software inspections Concerned with analysis of the static system representation to discover problems *(*static verification)
  - May be supplement by tool-based document and code analysis.

- Software testing Concerned with exercising and observing product behaviour (dynamic verification)
  - The system is executed with test data and its operational behaviour is observed.

Inspections and testing



Software inspections

- These involve people examining the source representation with the aim of discovering anomalies and defects.
- Inspections not require execution of a system so may be used before implementation.
- They may be applied to any representation of the system (requirements, design, configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.

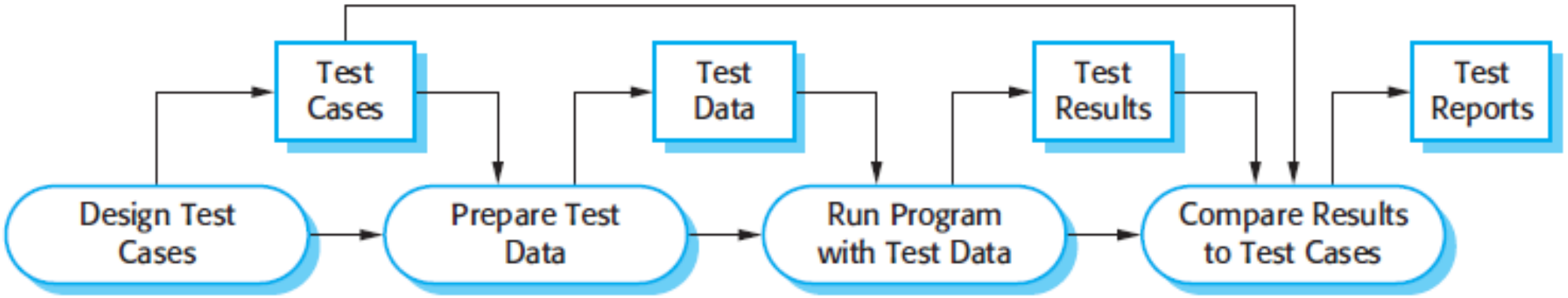
Advantages of inspections

- During testing, errors can mask (hide) other errors. Because inspection is a static process, you don’t have to be concerned with interactions between errors.
- Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available.
- As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.

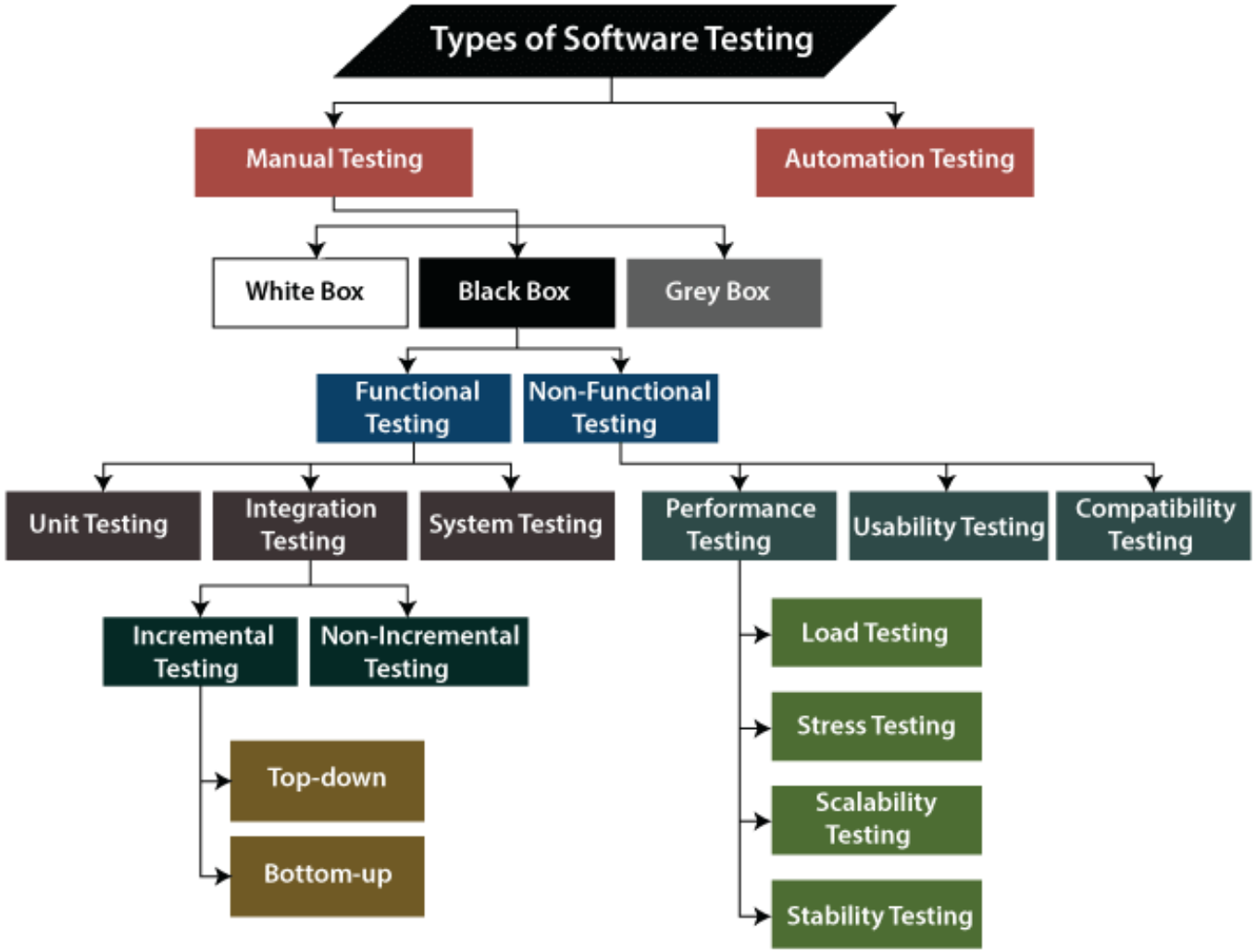
Inspections and testing

- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V & V process.
- Inspections can check conformance with a specification but not conformance with the customer’s real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

A model of the software testing process

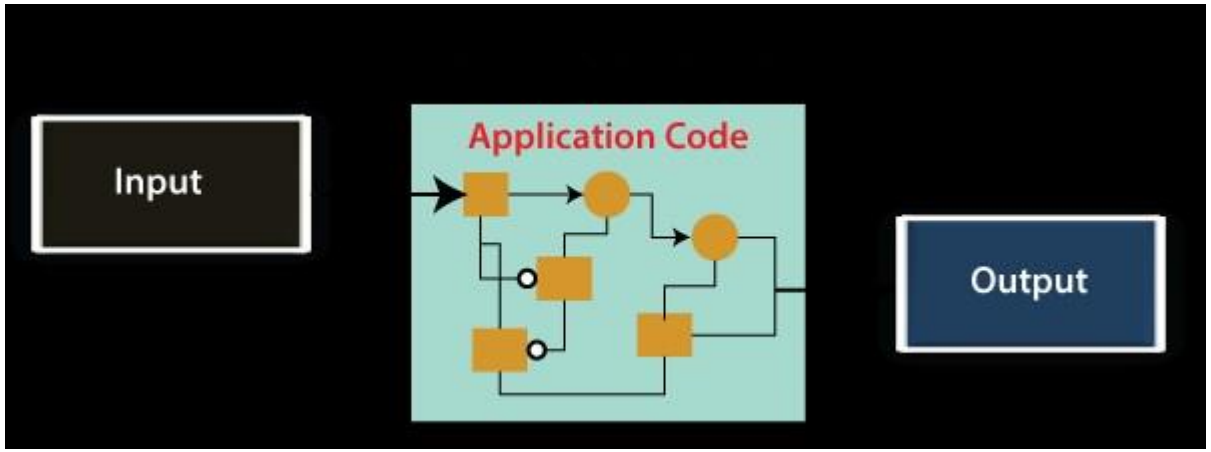


Types of Testing



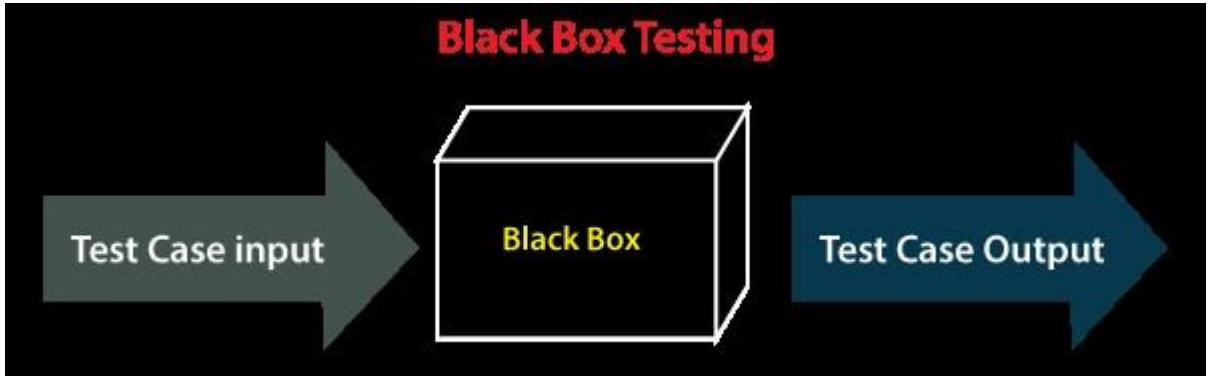
White Box Testing

- In white-box testing, the developer will inspect every line of code before handing it over to the testing team or the concerned test engineers.
- White box testing is also known as **open box testing, glass box testing, structural testing, clear box testing, and transparent box testing**.
  - Exercise all input and output parameters of each component.
  - Exercise all components and all calls (each component is called at least once and every component is called by all possible callers.)
  - Use conditional and iteration testing as in unit testing.



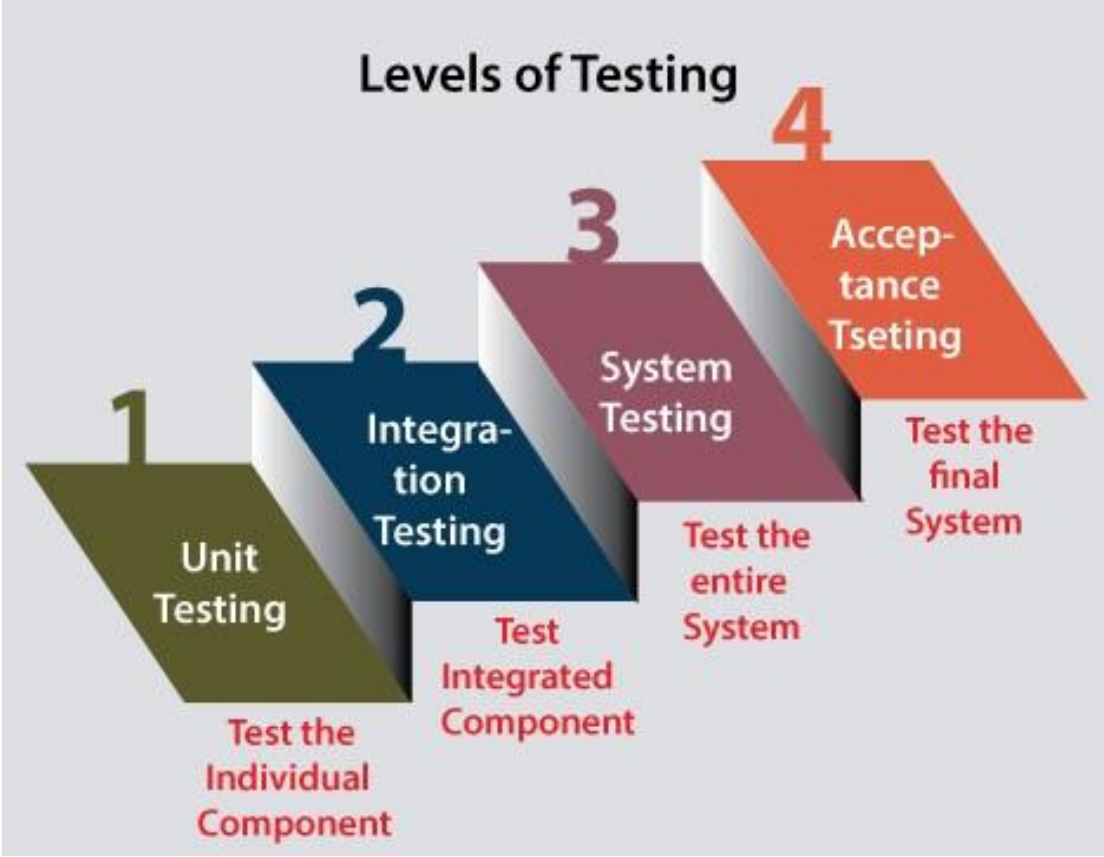
Black Box Testing

- Another type of manual testing is black-box testing. In this testing, the test engineer will analyze the software against requirements, identify the defects or bug, and sends it back to the development team.
- The source code is not visible in this testing; that's why it is known as **black-box testing**.



Stages of testing

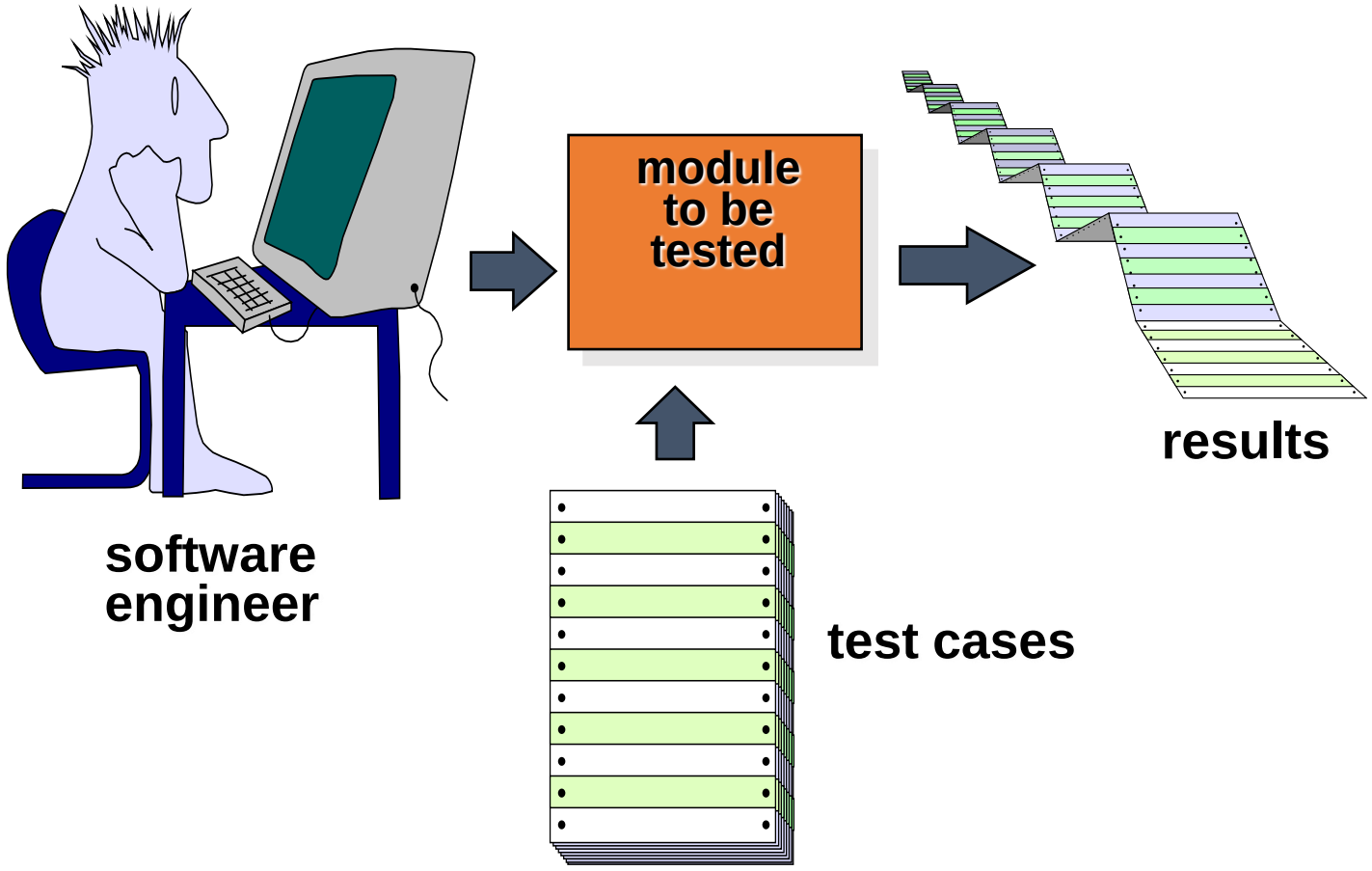
- Development testing, where the system is tested during development to discover bugs and defects.
- Release testing, where a separate testing team test a complete version of the system before it is released to users.
- User testing, where users or potential users of a system test the system in their own environment.



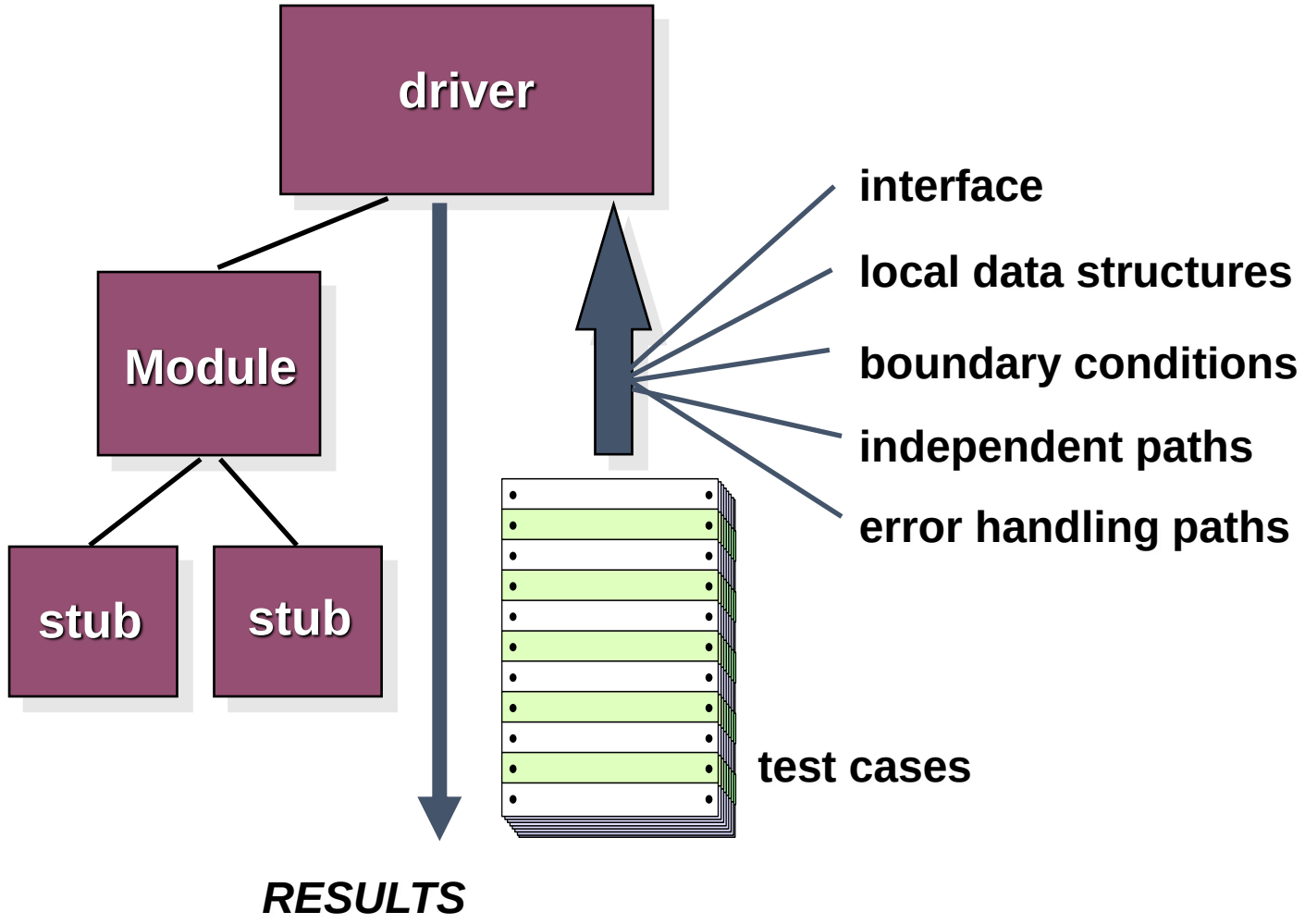
Development/ Functional testing

- The test engineer will check all the components systematically against requirement specifications is known as **functional testing**. Functional testing is also known as **Component testing**.
- In functional testing, all the components are tested by giving the value, defining the output, and validating the actual output with the expected value.
- The diverse types of Functional Testing contain the following:
  - Unit Testing
  - Integration Testing
  - System Testing

Unit Testing



Unit Test Environment



Unit testing

- Unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Units may be:
  - Individual functions or methods within an object
  - Object classes with several attributes and methods
  - Composite components with defined interfaces used to access their functionality.

Unit test effectiveness

- The test cases should show that, when used as expected, the component that you are testing does what it is supposed to do.
- If there are defects in the component, these should be revealed by test cases.
- This leads to 2 types of unit test case:
  - The first of these should reflect normal operation of a program and should show that the component works as expected.
  - The other kind of test case should be based on testing experience of where common problems arise. It should use abnormal inputs to check that these are properly processed and do not crash the component.

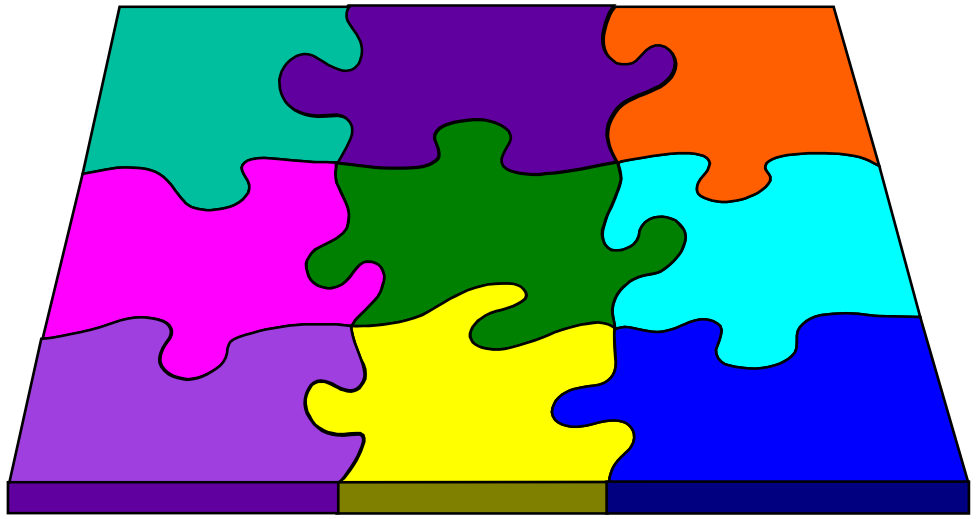
Integration Testing

- Once we are successfully implementing the unit testing, we will go integration testing. It is the second level of functional testing, where we test the data flow between dependent modules or interface between two features is called integration testing.
  - The entire system is viewed as a collection of subsystems (sets of classes) determined during the system and object design.
  - The order in which the subsystems are selected for testing and integration determines the testing strategy
- Types of Integration Testing
  - Incremental Testing
    - Bottom up integration
    - Top down integration
  - Non-Incremental Testing
    - Big bang integration

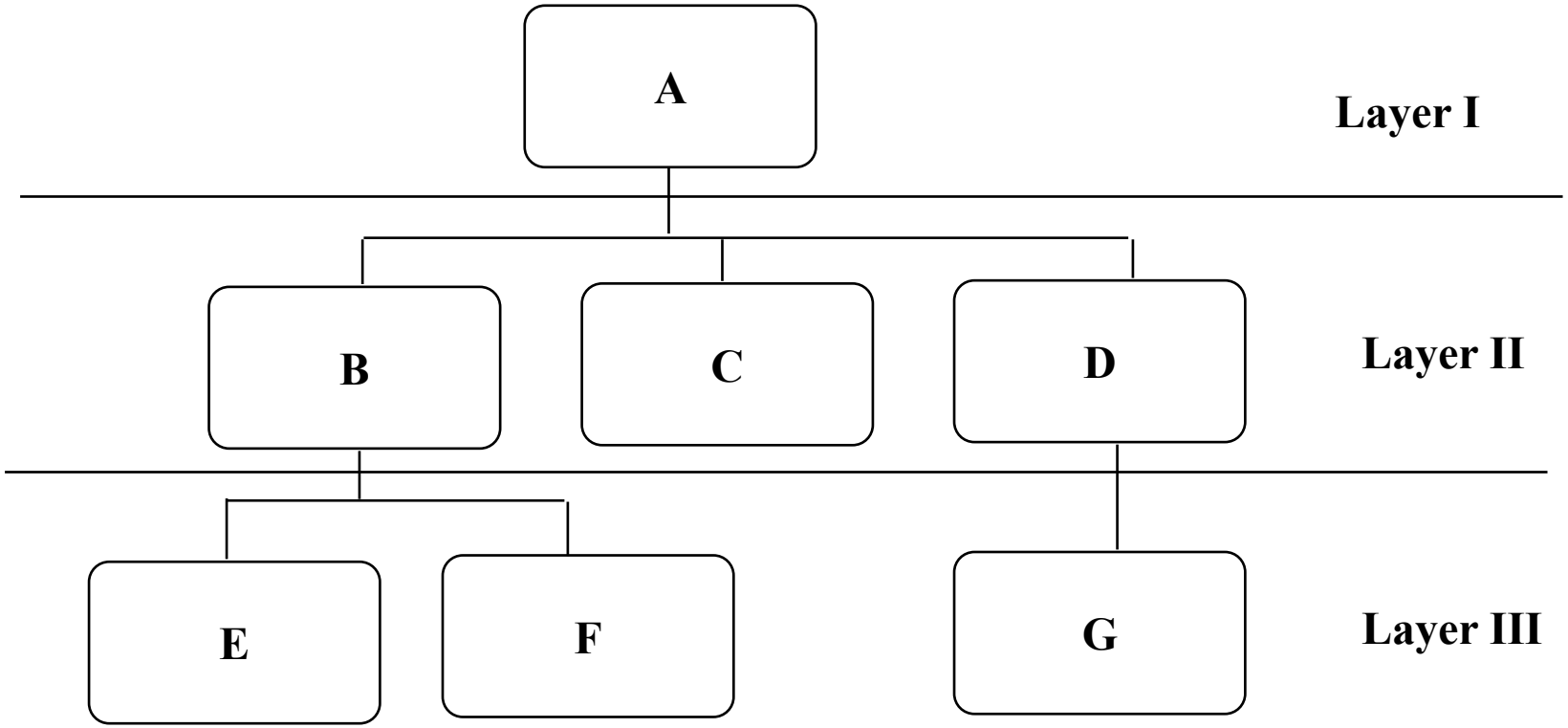
Integration Testing Strategies

Options:

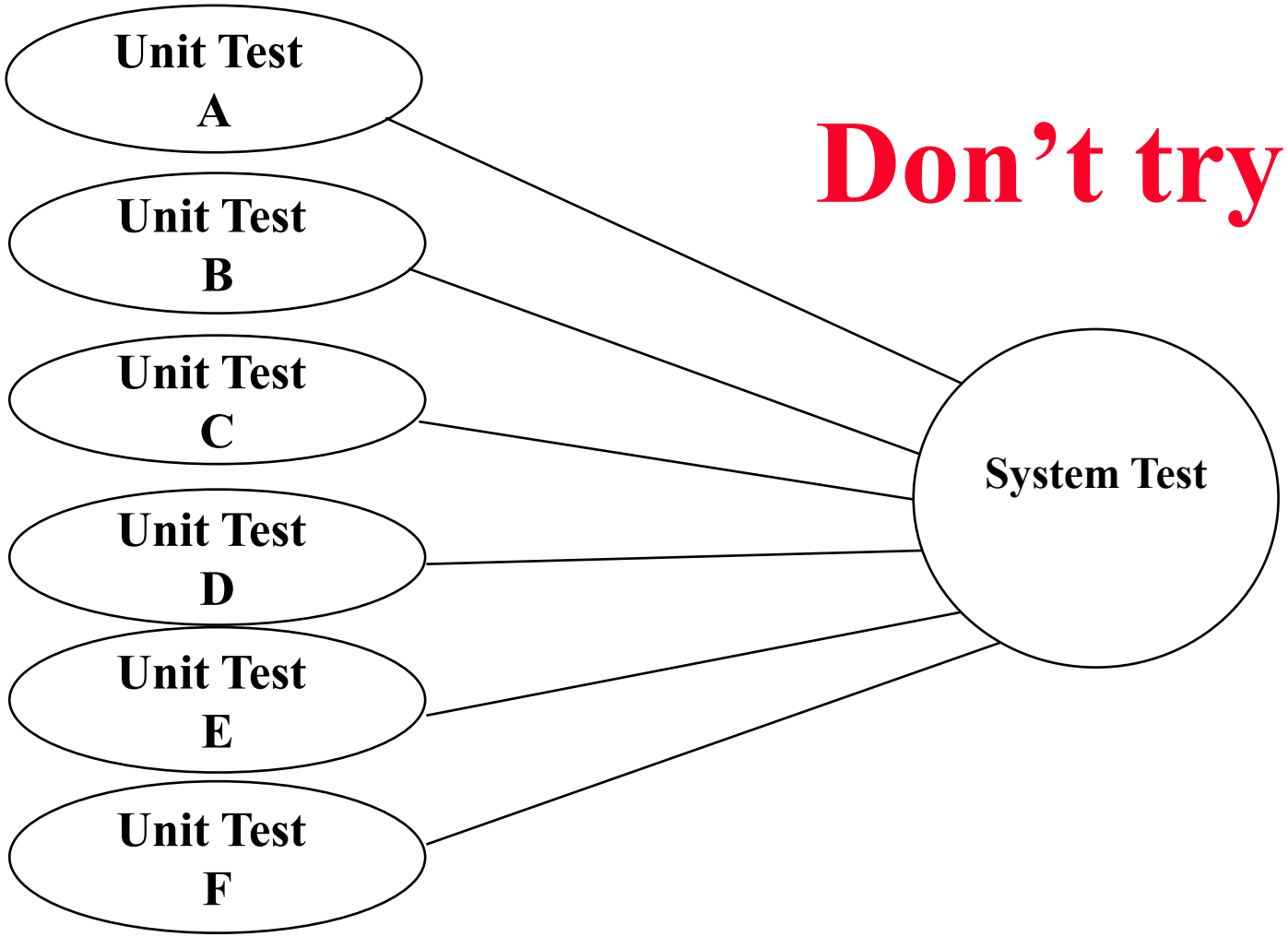
- **the “big bang” approach**
- **an incremental construction strategy**



Example: Three Layer Call Hierarchy

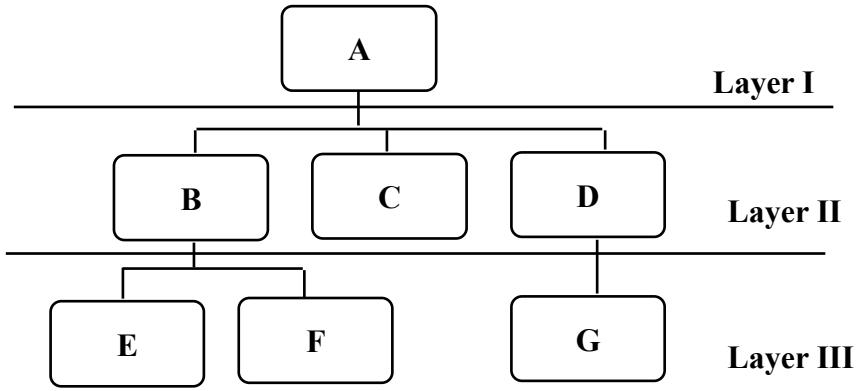


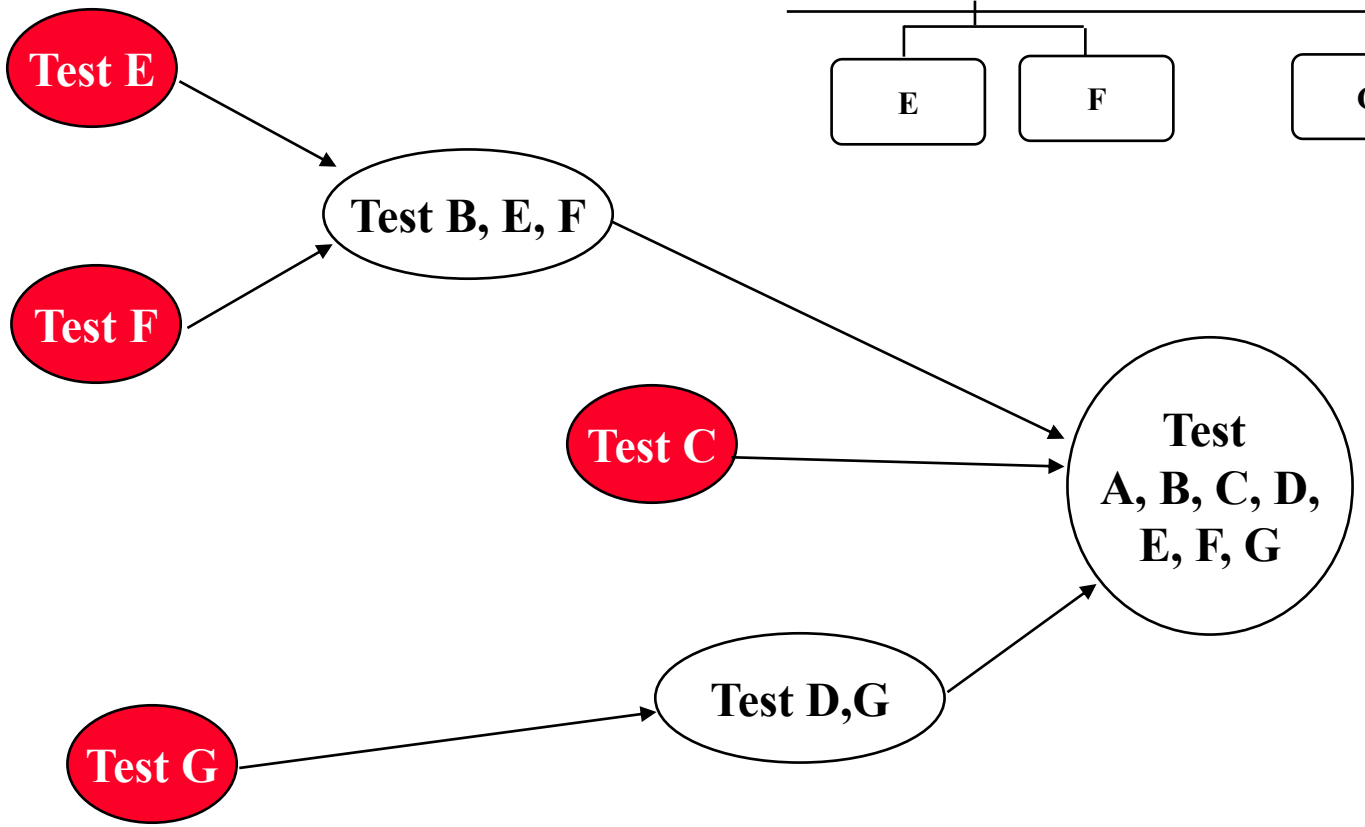
Integration Testing: Big-Bang Approach



Don’t try this!

Bottom-up Integration





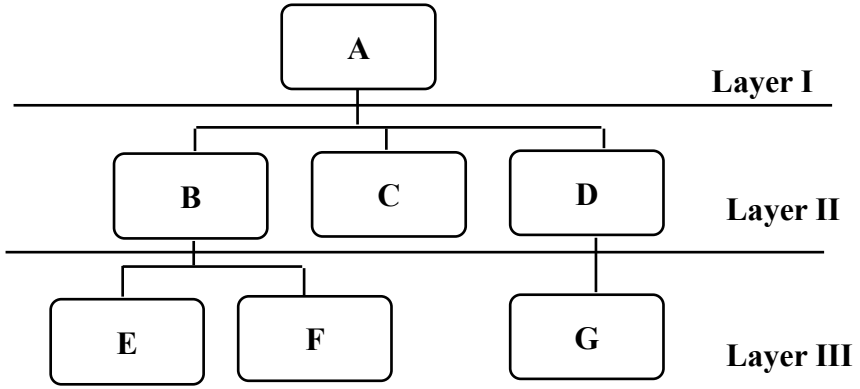
Special program needed to do the testing, Test Driver:

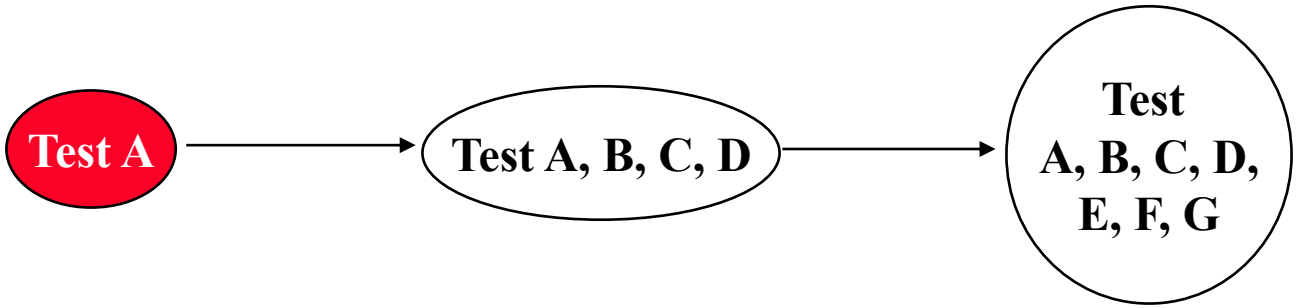
A routine that calls a subsystem and passes a test case to it

Pros and Cons of bottom up integration testing

- Bad for functionally decomposed systems:
  - Tests the most important subsystem (UI) last
- Useful for integrating the following systems
  - Object-oriented systems
  - real-time systems
  - systems with strict performance requirements

Top-down Integration Testing





Layer I

Layer I + II

All Layers

Special program is needed to do the testing, *Test stub :*

A program or a method that simulates the activity of a missing subsystem by answering to the calling sequence of the calling subsystem and returning back fake data.

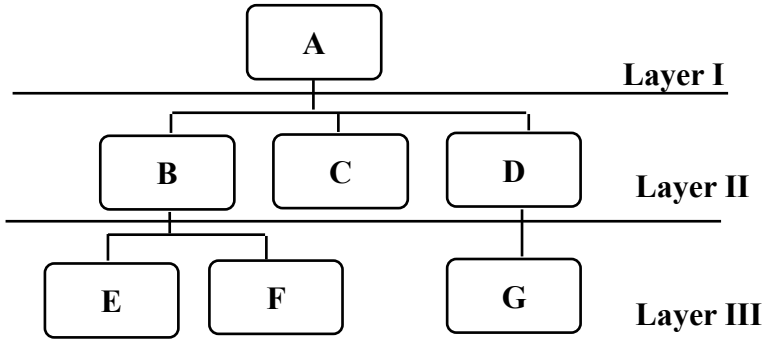
Pros and Cons of top-down integration testing

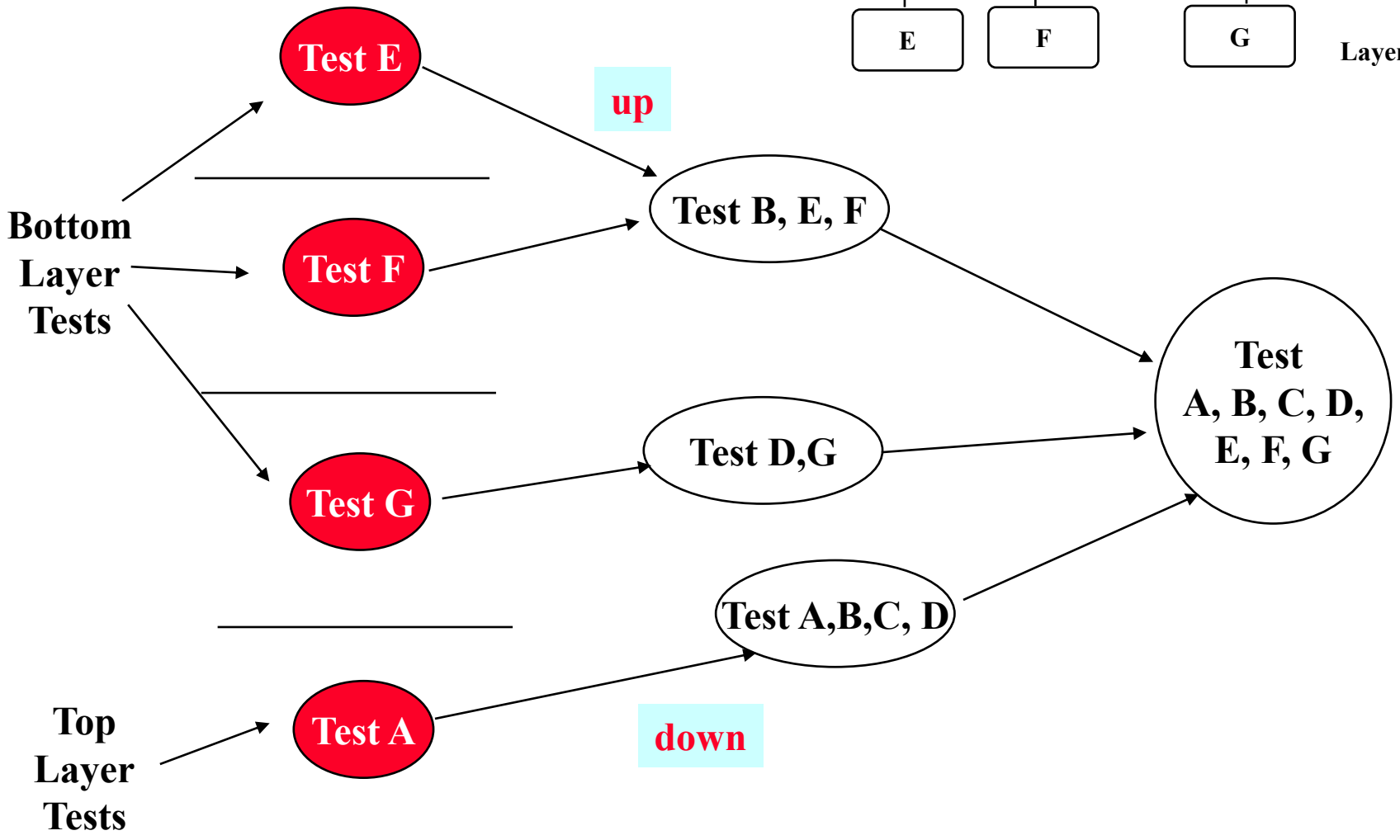
- Test cases can be defined in terms of the functionality of the system (functional requirements)
- *Writing stubs can be difficult*: Stubs must allow all possible conditions to be tested.
- Possibly a very large number of stubs may be required, especially if the lowest level of the system contains many methods.
- One solution to avoid too many stubs: *Modified top-down testing strategy*
  - Test each layer of the system decomposition individually before merging the layers
  - Disadvantage of modified top-down testing: Both, stubs and drivers are needed

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
- How do you select the target layer if there are more than 3 layers?
  - Heuristic: Try to minimize the number of stubs and drivers

Sandwich Testing Strategy





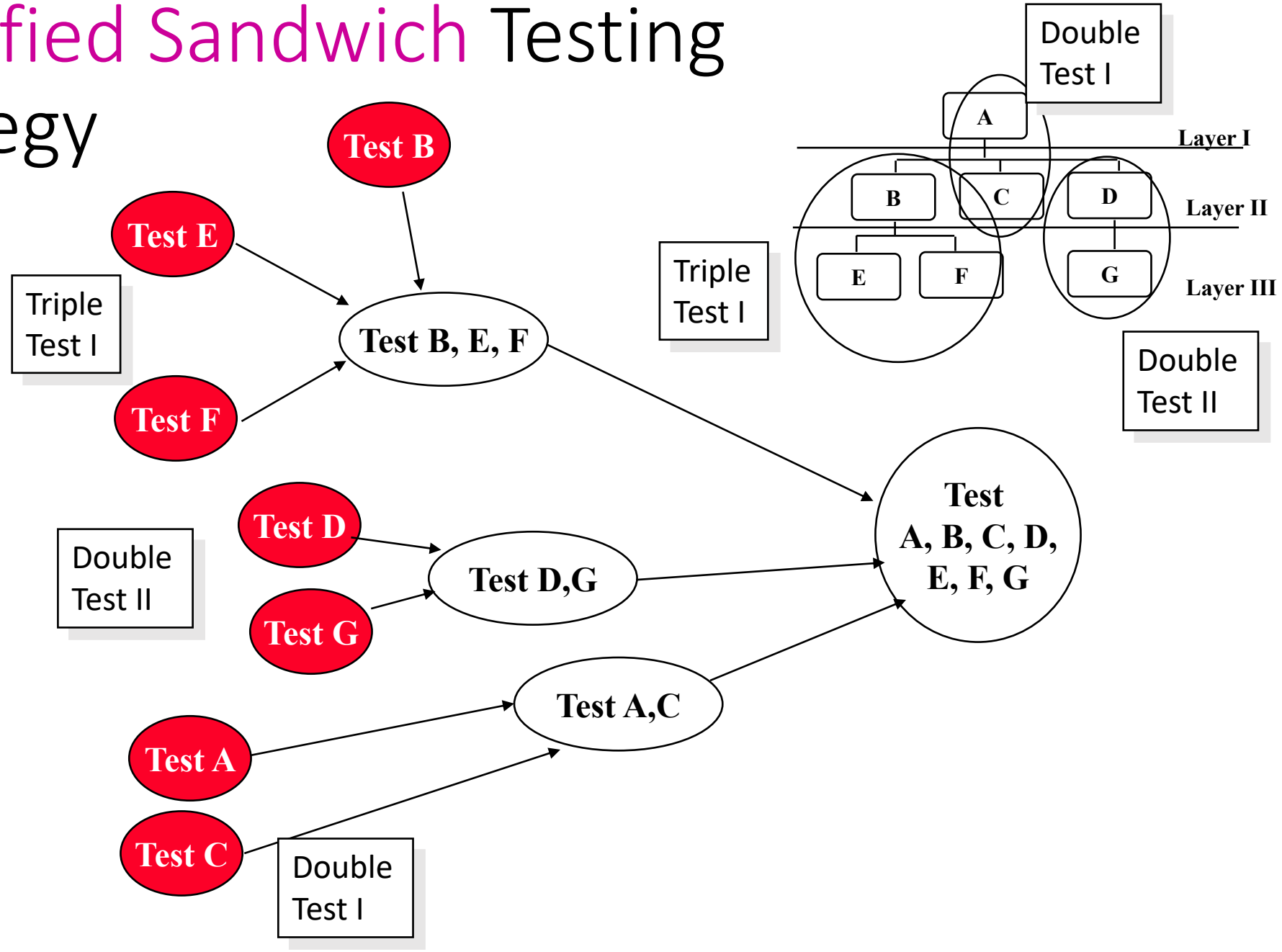
Pros and Cons of Sandwich Testing

- Top and Bottom Layer Tests can be done in ***parallel***
- Does not test the individual subsystems thoroughly before integration
- Solution: Modified sandwich testing strategy

Modified Sandwich Testing Strategy

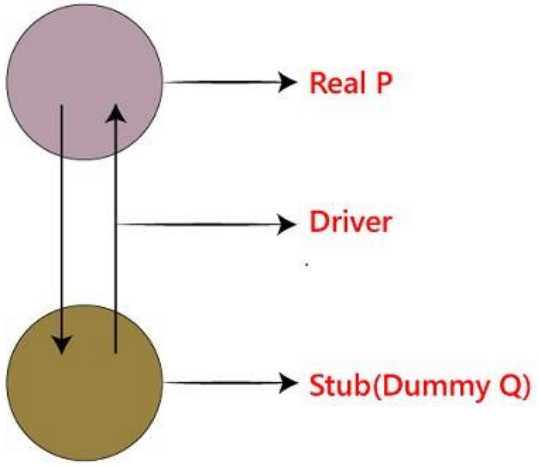
- Test in parallel:
  - Middle layer with *drivers and stubs*
  - Top layer with stubs
  - Bottom layer with drivers
- Test in parallel:
  - Top layer accessing middle layer (top layer replaces drivers)
  - Bottom accessed by middle layer (bottom layer replaces stubs)

Modified Sandwich Testing Strategy



Driver and Stub

- The stub is a dummy module that receives the data and creates lots of probable data, but it performs like a real module. When a data is sent from module P to Stub Q, it receives the data without confirming and validating it, and produce the estimated outcome for the given data.
- The function of a driver is used to verify the data from P and sends it to stub and also checks the expected data from the stub and sends it to P.
- The driver is one that sets up the test environments and also takes care of the communication, evaluates results, and sends the reports. We never use the stub and driver in the testing process.

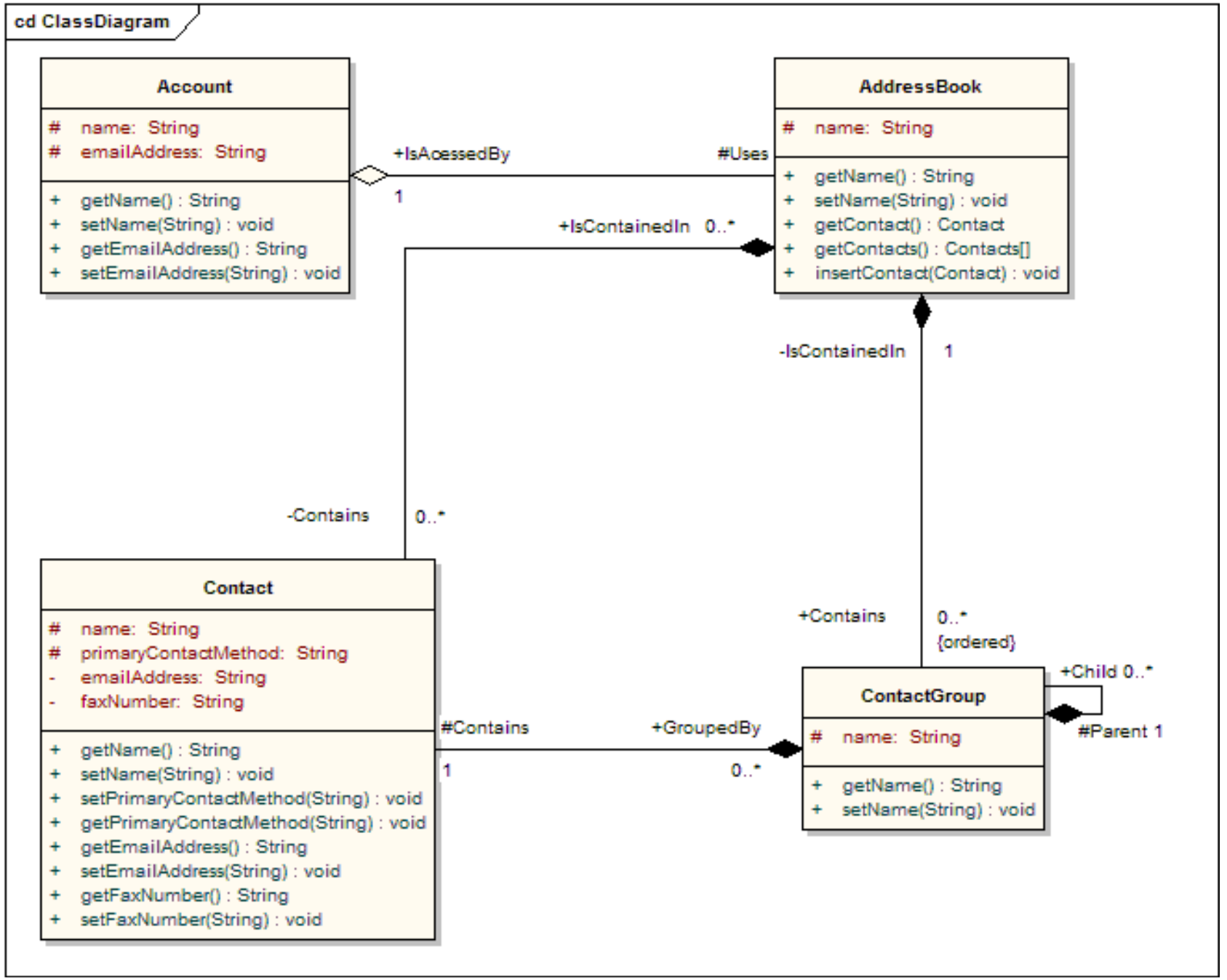


Object-Oriented Testing

- 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

Testing the CRC Model

1. Revisit the CRC model and the object-relationship model.
2. Inspect the description of each CRC index card to determine if a delegated responsibility is part of the collaborator's definition.
3. Invert the connection to ensure that each collaborator that is asked for service is receiving requests from a reasonable source.
4. Using the inverted connections examined in step 3, determine whether other classes might be required or whether responsibilities are properly grouped among the classes.
5. Determine whether widely requested responsibilities might be combined into a single responsibility.
6. Steps 1 to 5 are applied iteratively to each class and through each evolution of the analysis model.


cd ClassDiagram
Account
name: String
emailAddress: String
+ getName() : String
+ setName(String) : void
+ getEmailAddress() : String
+ setEmailAddress(String) : void
+IsAcessedBy
1
#Uses
AddressBook
name: String
+ getName() : String
+ setName(String) : void
+ getContact() : Contact
+ getContacts() : Contacts[]
+ insertContact(Contact) : void
+IsContainedIn 0..*
-IsContainedIn
1
-Contains
0..*
Contact
name: String
primaryContactMethod: String
- emailAddress: String
- faxNumber: String
+ getName() : String
+ setName(String) : void
+ setPrimaryContactMethod(String) : void
+ getPrimaryContactMethod(String) : void
+ getEmailAddress() : String
+ setEmailAddress(String) : void
+ getFaxNumber() : String
+ setFaxNumber(String) : void
#Contains
1
+GroupedBy
0..*
+Contains
0..*
{ordered}
ContactGroup
name: String
+ getName() : String
+ setName(String) : void
+Child 0..*
#Parent 1

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

System testing

- Done as the Development progresses
- Users Stories, Features and Functions are Tested
- Testing done in Production Environment
- Quality Tests are executed (Performance, Reliability, etc.)
- Defects are reported
- Tests are automated where possible

Requirements based testing

- Requirements-based testing involves examining each requirement and developing a test or tests for it.
- MHC-PMS requirements:
  - If a patient is known to be allergic to any particular medication, then prescription of that medication shall result in a warning message being issued to the system user.
  - If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.

Requirements tests

- Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system.
- Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.
- Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.
- Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.
- Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

Features tested by scenario

- Authentication by logging on to the system.
- Downloading and uploading of specified patient records to a laptop.
- Home visit scheduling.
- Encryption and decryption of patient records on a mobile device.
- Record retrieval and modification.
- Links with the drugs database that maintains side-effect information.
- The system for call prompting.

A usage scenario for the MHC-PMS

Kate is a nurse who specializes in mental health care. One of her responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side -effects.
On a day for home visits, Kate logs into the MHC-PMS and uses it to print her schedule of home visits for that day, along with summary information about the patients to be visited. She requests that the records for these patients be downloaded to her laptop. She is prompted for her key phrase to encrypt the records on the laptop.

One of the patients that she visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side -effect of keeping him awake at night. Kate looks up Jim's record and is prompted for her key phrase to decrypt the record. She checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so she notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. He agrees so Kate enters a prompt to call him when she gets back to the clinic to make an appointment with a physician. She ends the consultation and the system re-encrypts Jim's record.

After, finishing her consultations, Kate returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for Kate of those patients who she has to contact for follow-up information and make clinic appointments.

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 is the initial testing process exercised to check whether the software under test is ready/stable for further testing.
- The term ‘**Smoke Testing**‘ comes from the hardware testing, in the hardware testing initial pass is done to check if it did not catch the fire or smoke in the initial switch on.

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.

Sanity Testing

- Sanity testing is performed on stable builds and it is also known as a variant of regression testing.
- Sanity testing was performed when we are receiving software build (with minor code changes) from the development team. It is a checkpoint to assess if testing for the build can proceed or not.
- In other words, we can say that sanity testing is performed to make sure that all the defects have been solved and no added issues come into the presence because of these modifications.
- Attributes
  - a **narrow and deep** method where limited components are protected deeply.
  - It is a subdivision of **regression testing**, which mainly emphases on the less important unit of the application.
  - Unscripted
  - Not documented
  - Performed by testers

Non-Functional Testing

- The next part of black-box testing is non-functional testing. It provides detailed information on software product performance and used technologies.
- Non-functional testing will help us minimize the risk of production and related costs of the software.
- Non-functional testing is a combination of performance, load, stress, usability and, compatibility testing.

Non-Functional Testing

- Stress Testing
  - Stress limits of system (maximum # of users, peak demands, extended operation)
- Volume testing
  - Test what happens if large amounts of data are handled
- Configuration testing
  - Test the various software and hardware configurations
- Compatibility test
  - Test backward compatibility with existing systems
- Security testing
  - Try to violate security requirements
- Timing testing
  - Evaluate response times and time to perform a function
- Environmental test
  - Test tolerances for heat, humidity, motion, portability
- Quality testing
  - Test reliability, maintain- ability & availability of the system
- Recovery testing
  - Tests system's response to presence of errors or loss of data.
- Human factors testing
  - Tests user interface with user

Test Cases for Performance Testing

- **Push the (integrated) system to its limits.**
- **Goal: Try to break the subsystem**
- **Test how the system behaves when overloaded.**
  - Can bottlenecks be identified? (First candidates for redesign in the next iteration
- **Try unusual orders of execution**
  - Call a receive() before send()
- **Check the system's response to large volumes of data**
  - If the system is supposed to handle 1000 items, try it with 1001 items.
- **What is the amount of time spent in different use cases?**
  - Are typical cases executed in a timely fashion?

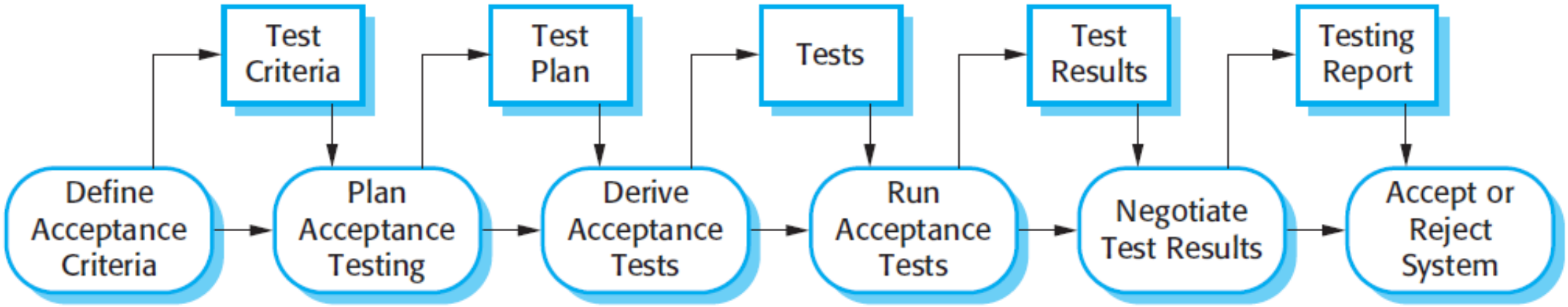
User testing

- User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.
- User testing is essential, even when comprehensive system and release testing have been carried out.
  - The reason for this is that influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

Types of user testing

- Alpha testing
  - Users of the software work with the development team to test the software at the developer's site.
- Beta testing
  - A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
- Acceptance testing
  - Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

The acceptance testing process



Stages in the acceptance testing process

- Define acceptance criteria
- Plan acceptance testing
- Derive acceptance tests
- Run acceptance tests
- Negotiate test results
- Reject/accept system

Gray Box Testing

- Another part of manual testing is Grey box testing. It is a collaboration of black box and white box testing.
- Since, the grey box testing includes access to internal coding for designing test cases. Grey box testing is performed by a person who knows coding as well as testing.
- In other words, we can say that if a single-person team done both white box and black-box testing, it is considered grey box testing.

Automation Testing

- The most significant part of Software testing is Automation testing. It uses specific tools to automate manual design test cases without any human interference.
- Automation testing is the best way to enhance the efficiency, productivity, and coverage of Software testing.
- It is used to re-run the test scenarios, which were executed manually, quickly, and repeatedly.
- In other words, we can say that whenever we are testing an application by using some tools is known as automation testing.

Final Thoughts

- *Think* -- before you act to correct
- Use tools to gain additional insight
- If you’re at an impasse, get help from someone else
- Once you correct the bug, use regression testing to uncover any side effects

Reference

1. Roger S. Pressman, “Software Engineering: A Practitioner’s Approach”, 7th edition, Chapter 17: Software Testing Strategies.
2. Ian Sommerville, “software Engineering”, 9th edition. Chapter 8: Software Testing.
3. https://www.javatpoint.com/