Software Testing & Quality Assurance

Objectives

To provide learner with knowledge in Software Testing techniques.

To understand how testing methods can be used as an effective tools in providing quality assurance concerning for software.

To provide skills to design test case plan for testing software

Expected Learning Outcomes

1. Understand various software testing methods and strategies.
2. Understand a variety of software metrics, and identify defects and managing those defects for improvement in quality for given software.
3. Learn to Design the test cases.
4. Learning the testing framework and understand the role of tester

Unit-I

Software testing strategies:

Strategic approach to software testing, Strategic issues, Unit Testing- Levels, techniques, tools [A], Integration testing [A], Validation testing, System testing, The art of debugging.

Procedural Design and use of Reusable components:

Procedural design, Structured design, Reusable code, Component based software engineering, Program verification.

Implementation of testing framework using java programming language through Junit, compare features of NUnit, Junit, TestNG, Mockito, PHPUnit.

15L

Unit 2

Technical metrics for software:
Software Quality, A framework for Technical software metrics, Metrics for the analysis model,
Metric for the design model, Metric for source code, Metric for testing, Metric for maintenance.
Software Reuse:
Management issues, The reuse process, Domain engineering, Building reusable components, Classifying and retrieving the components, Economics of software reuse.
Selenium testing framework:
Selenium Webdriver Script: JAVA Code Example[A] , selenium and automation test tools, TestNG in Eclipse for Selenium WebDriver[A], Annotations, Framework [A], Examples in Selenium, Use of JXL API in selenium.
15L

Unit 3

Client server software engineering:

The structure of client /server systems, Software engineering for client server systems

Quality Improvement :

Pareto Diagrams, Cause-effect Diagrams, Scatter Diagrams, Run charts

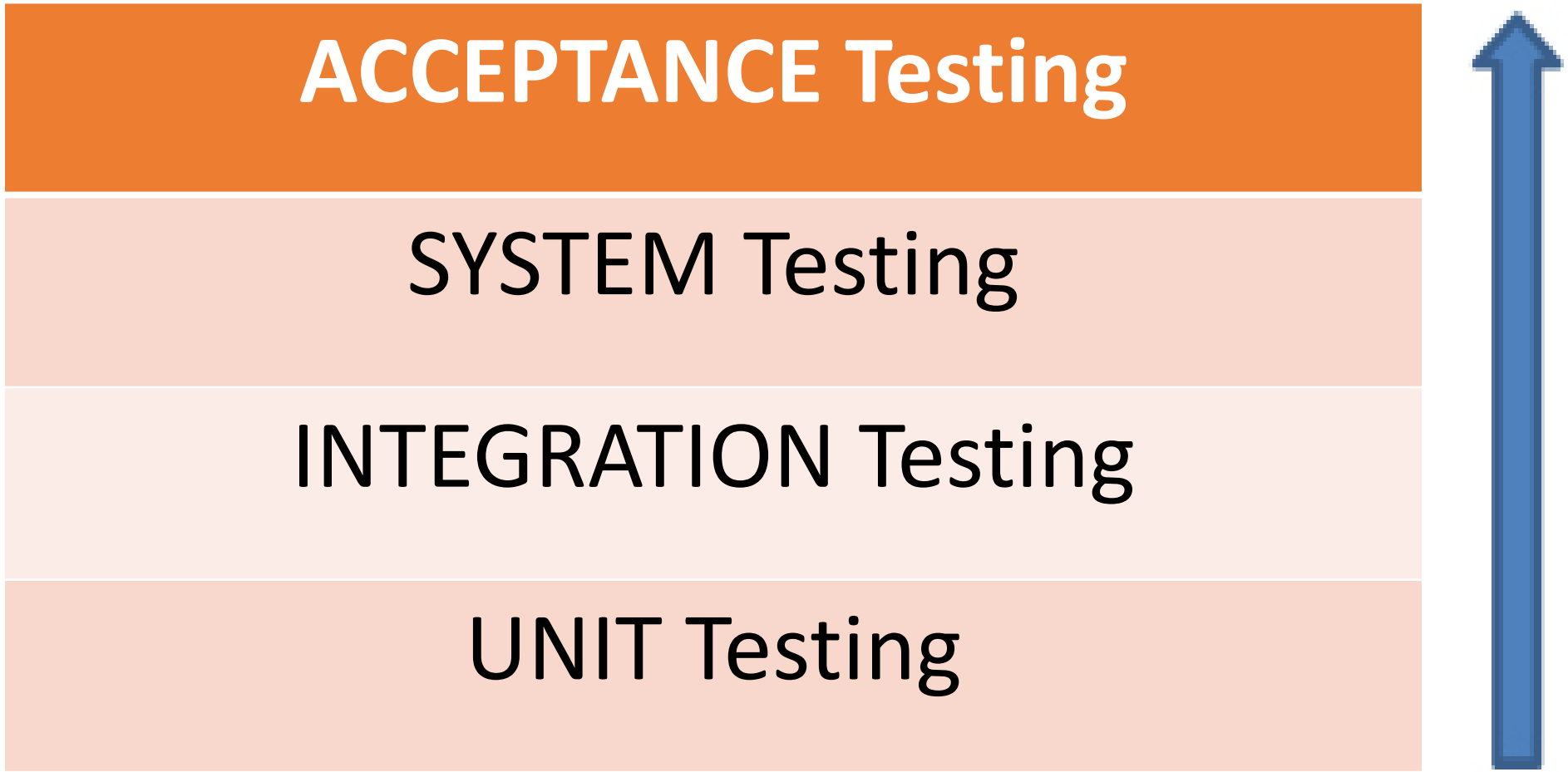
Quality Costs :

Defining Quality Costs, Types of Quality Costs, Quality Cost Measurement, Utilizing Quality Costs for Decision-MakingMaven & Jenkins Integration with Selenium, Importance of Maven and using with TestNG Selenium, Jenkins and configure it to Run Maven with TestNg Selenium, Scheduling Jenkins for automatic execution, Benefits of Jenkins

Why Unit Testing?

Unit testing represents first level of testing followed by integration and other levels of the testing. It uses modules for testing and thereby reducing the dependency on :

- ✓ Testing frameworks
- ✓ Stubs
- ✓ Drivers



Levels of Testing

Unit testing helps in:

- Finding errors and defects.
- Handling functional correctness
- Corelating input and output values.
- Optimizing algorithm

Unit Testing Techniques:

The process of unit testing can be executed with three important testing techniques, which are:

1. **Structural Technique:**

Unit testing uses white box testing techniques as it makes use of code in application and uses system's structure and internal implementation for designing and creating test cases :

- ✓ Data flow Testing
- ✓ Control Flow Testing
- ✓ Branch Coverage Testing
- ✓ Statement Coverage Testing
- ✓ Decision Coverage Testing

2. **Functional Testing Technique**:

Used to validate the functionality of the application, and represents a black box testing technique involving:

- Input domain testing.
- Boundary Value.
- Syntax Checking.
- Equivalence Partitioning.

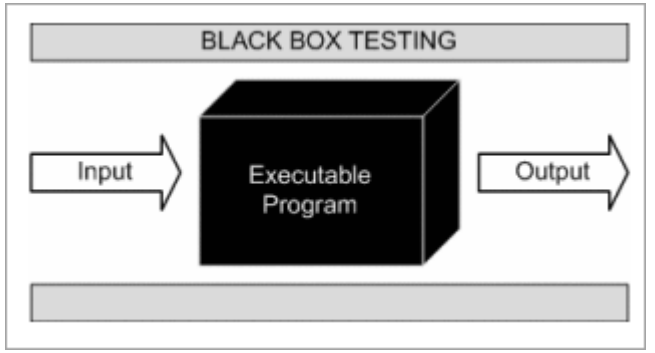
3. **Error Based Techniques**:

Allows the user to identify bugs in the application. It uses:

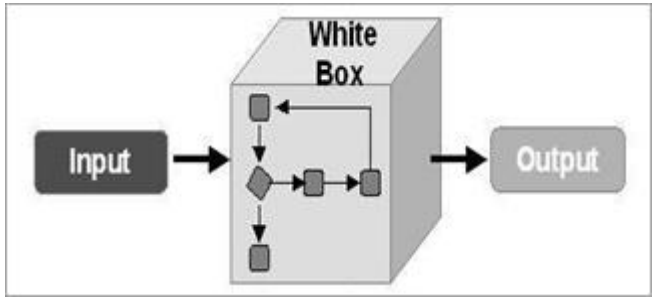
- Fault Seeding.
- Mutation Testing.

Types of Unit Testing:

a.Black Box Testing: Enables testing the user interface of the application along with input and output.



b.White Box Testing: Enables testing the functional behavior of the application and to validate their execution.



c.Gray Box Testing: Enables executing test suite and test cases.

Popular **unit testing tools** used in IT industry are:

1.JUnit.

2.NUnit.

3.JMockit.

4.Selenium.

5.EMMA.

6.Quilt HTTP.

7.HTMLUnit.

8.PHPUnit.

1.Junit: used for Java programming language. **Uses assertions to identify test method.**

2.Nunit : used for .net languages. **Enables data-driven tests which can run in parallel.**

3.JMockit : open source Unit testing tool. **Enables Line coverage, Path Coverage, and Data Coverage.**

4. EMMA: Used for **analyzing code** written in Java language. Allows coverage types like method, line, basic block. Being Java-based so it is without external library dependencies and can access the source code.

5. PHPUnit: Used for PHP . It takes small portions of code which is called **units and test each of them separately**. Allows developer to **use pre-define assertion methods** to assert that a system behave in a certain manner.

INTEGRATION TESTING

- Integration Testing combines different modules in the application and test as a whole. [Individual units]
- Integration testing checks integration between software modules.
- Integration Testing starts with interface specification. [Module]

- Performed after unit testing and before system testing.

  [Anytime]

- Integration Testing is performed by the testing team.

  [Developer]

- Integration Testing is a kind of black-box testing.

  [WhiteBox]

- Integration tests verify that your code works with external dependencies correctly.

Stubs and Drivers :

Dummy programs in Integration testing used to facilitate the software testing activity.

These programs act as a substitutes for the missing modules in the testing.

They do not implement the entire programming logic of the software module but they simulate data communication with the calling module during testing.

Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

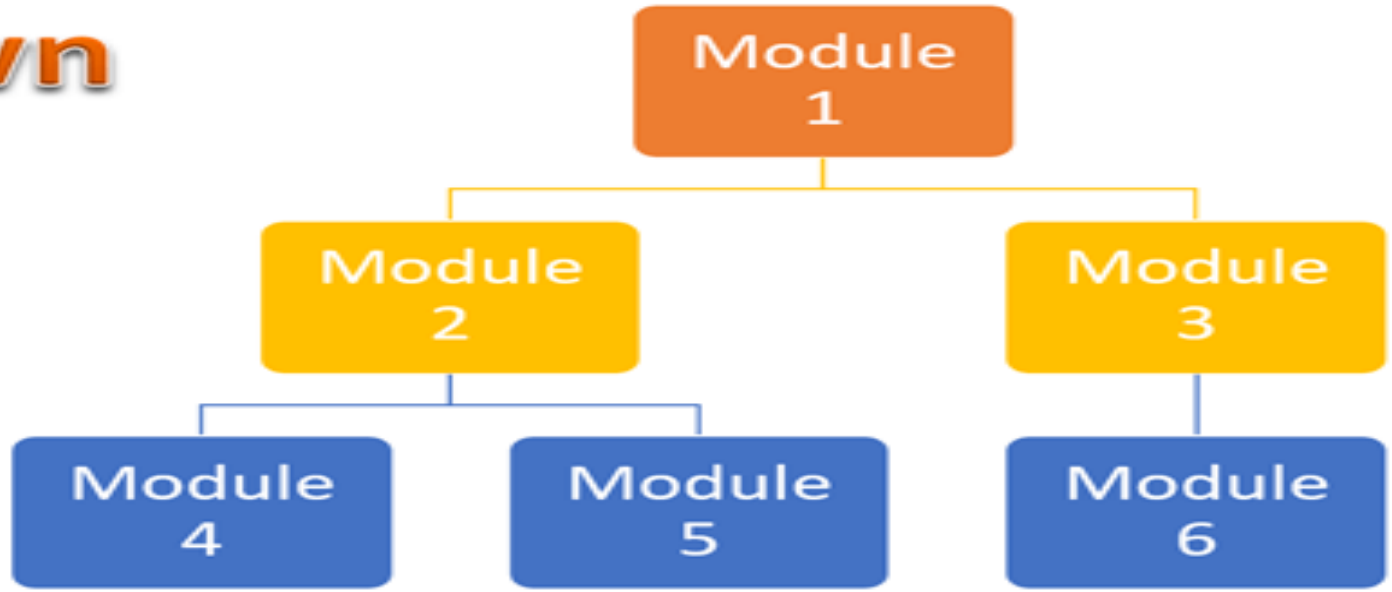
Stubs	Driver
✓ Used in Top down approach	**Used in Bottom up approach**
✓ Top most module is tested first	**Lowest modules are tested first.**
✓ Dummy program of lower level components	**Dummy program for Higher level component**

APPROACH TO INTEGRATION TESTING

Top Down

Hybrid

Bottom Up



Eg. Of Integration Test cases for Linkedin application:

•Verifying the interface link between the login page and the home page

•Verifying the interface link between the home page and the profile page

•Verify the interface link between the network page and connection pages i.e. clicking accept button on Invitations of the network page should show the accepted invitation in your connection page once clicked.

•Verify the interface link between the Notification pages and subsequent action

Validation Ensures that:

Software Being Developed or Changed
Satisfies Functional and Other Requirements

Validation comes Into Picture
at the End of the Development Cycle

- **It Views the Complete System Exactly Opposite of Verification**

- **It Focuses on Smaller Sub-Systems**

What are the Techniques of Validation Testing ?

Formal Methods :
A Technique of Validation Testing

- **Involves use of Mathematical and Logical Techniques to: Express, Investigate, & Analyze the Specification, Design, Documentation and Behavior of Hardware as well as Software**

Fault Injection :

Is an Intentional Activation of Faults by Either Hardware or Software to Observe the System Operation under such Faulty Situations

Hardware Fault Injection :

Also known as Physical Fault Injection since Faults are Injected into the Physical Hardware

Software Fault Injection :
A Technique of Validation Testing

- **Involves Injection of Errors into the Computer Memory through some Software Techniques**

- **It is a sort of a Simulation of Hardware Fault Injection**

Dependency Analysis :
A Technique of Validation Testing

Involves Identification of Hazards and Subsequently Proposing Methods to Reduce the Risk of the Hazards

Hazard Analysis :
A Technique of Validation Testing

Involves using Instructions to Identify Hazards, Their Root Causes, and Possible Countermeasures

Risk Analysis :
A Technique of Validation Testing

Goes Beyond Hazard Analysis by Identifying the Possible Consequences of Each Hazard and Their Probability of Occurrence

System Testing

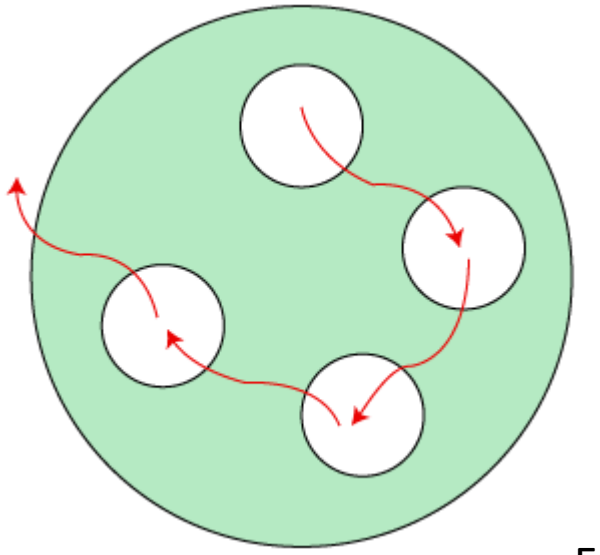
System Testing:

Carried out by developers

Goal: Determine if the system meets the *requirements* (functional and *global*)

Terminology:
system testing here = validation testing

- Types of system testing
  - UI testing
  - build/check-in tests
  - load testing
  - usability testing
  - security testing
  - monitoring
  - performance profiling



End to end

Structured design :

- Based on '**divide and conquer**' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.
- The small pieces of problem are solved by means of **solution modules.**
- Structured design emphasis that these modules be **well organized** in order to achieve precise solution.
- These modules are **arranged in hierarchy.**

structured design uses following principles to communicate among multiple modules:-

- **Cohesion** - grouping of all functionally related elements.
- **Coupling** - communication between different modules.
- Structured design should have high cohesion and low coupling arrangements.

- ✓ *High cohesion means* ***keeping parts of a code base that are related to each other in a single place.***
- ✓ *Low coupling, is about* ***separating unrelated parts of the code base as much as possible.***
- ✓ **Ideal** is the code that follows the guideline: loosely coupled and highly cohesive.
- ✓ high cohesion and high coupling :is an anti-pattern and basically stands for a single piece of code that does all the work at once

Procedural design :

- is used to **model programs** that have flow of data from input to output.
- It represents the **architecture of a program** as a set of interacting processes that pass data from one to another.
- Design Tools :The two diagramming tools used in procedural design are **data flow diagrams** and **structure charts.**

The DFD is a conceptual model –

- It doesn't represent the computer program, it represents what the program must accomplish.
- By showing the input and output of each major task, it shows how **data must move through and be transformed by the program.**

DFD purpose: show the scope and boundaries of a system as a whole.

It begins with a **context diagram as level 0** of the DFD diagram, a simple representation of the whole system.

To elaborate further from that, we drill down to a **level 1 diagram with lower-level functions** decomposed from the major functions of the system.

Further to evolve to a **level 2 diagram** when further analysis is required.

Factoring is the second phase of procedural design in which we create a structure chart that shows what program components need to be implemented.

This is implemented in two passes.

First, arrange DFD hierarchically.

Second, identify exactly which conceptual processes are to be implemented as physical components in the program

COMPONENT-BASED SOFTWARE ENGINEERING

- **Component Based Software Engineering (CBSE)** is a process for design and development of computer-based systems using reusable software components.
- It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific.

- Apart from reuse, CBSE is based on software engineering design principles:
  - Components are independent so do not interfere with each other;
  - Component implementations are hidden;
  - Communication is through well-defined interfaces;
  - Component platforms are shared and reduce development costs.

- Components provide a service irrespective of component execution or its programming language
  - A component is an independent executable entity that can be made up of one or more executable objects;
  - The component interface is published and all interactions are through the published interface;

Characteristics:

- Components are deployable entities.
- Components do not define types.
- Component implementations are opaque.
- Components are language-independent.
- Components are standardised.
- Is a binary component, not to be compiled before deployment

- Examples of component models
  - EJB model (Enterprise Java Beans)
  - COM model (.NET model)
  - Corba Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.

An interface is not a class.

- Although an *instance of a class can be created* (instantiated) to form a COM component, an interface cannot be instantiated by itself because it carries no implementation.
- A COM component must implement that interface and that COM component must be instantiated in order for an interface to exist.

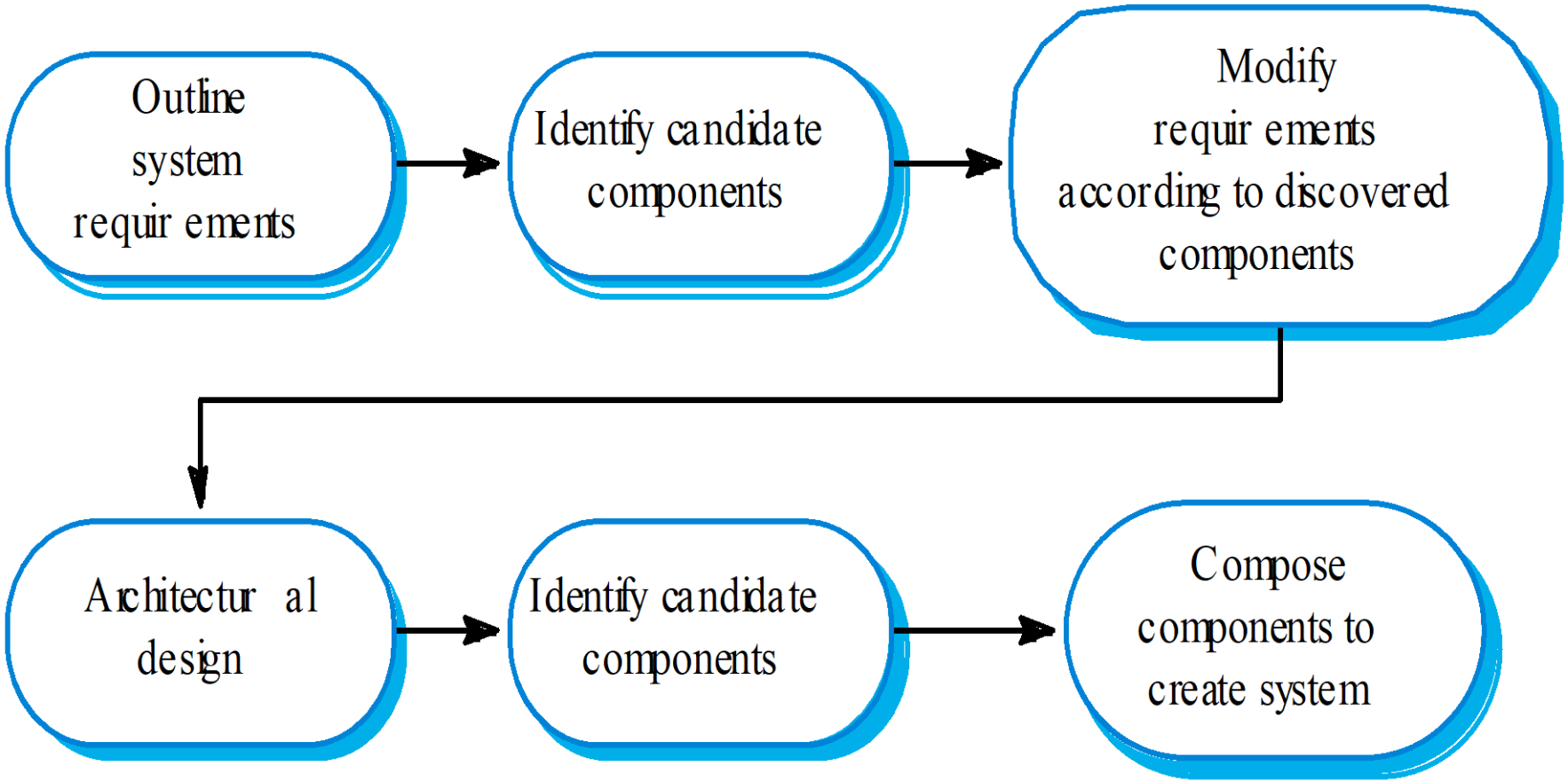
Interfaces are strongly typed.

- Every interface has its own interface identifier, thereby eliminating any chance of collision that would occur with human-readable names.
- The difference between components and interfaces has two important implications:
- When we create a new interface, we must also create a new identifier for that interface.
- When we use an interface, we must use the identifier for the interface to request a pointer to the interface.

Interfaces are immutable.

- ✓ COM interfaces are never versioned, which means that version conflicts between new and old components are avoided.
- ✓ A new version of an interface, created by adding more functions or changing semantics, is an entirely new interface and is assigned a new, unique identifier.

CBSE process



Types of composition

- Sequential composition : where the composed components are executed in sequence.
- Hierarchical composition: where one component calls on the services of another.
- Additive composition : where the interfaces of two components are put together to create a new component.

What is a Test Framework?

A testing framework is a set of guidelines or rules used for creating and designing test cases. It results in :

- Improved test efficiency
- Minimal manual intervention
- Maximum test coverage
- Reusability of code

Framework can be defined as a set of guidelines which when followed produces beneficial results.

Test Automation Frameworks:

1. Linear Scripting – Record & Playback

A Testing Automation Frameworks and also known as **“Record & Playback”**.

In this Automation Testing Framework, Tester manually records each step (Navigation and User Inputs), Inserts Checkpoints (Validation Steps) in the first round and then , Plays back the recorded script in the subsequent rounds.

- The scripts developed using this framework aren’t reusable. The data is hardcoded into the test script, meaning the test cases cannot be re-run with multiple sets and will need to be modified if the data is altered.

2. Modular Based Testing Framework

Implementing a modular framework will require testers to divide the application under test into **separate units, functions, or sections,** each of which will be tested in isolation.

After breaking down the application into individual modules, a test script is created for each part and then combined to build larger tests in a hierarchical fashion. These larger sets of tests will begin to represent various test cases.

Emphasis in using the modular framework is to build an abstraction layer, so that any changes made in individual sections won't affect the overall module.

Features of JUnit

- JUnit is an open source framework, which is used for writing and running tests.
- Provides annotations to identify test methods.
- Provides assertions for testing expected results.

JUnit provides "**assert" commands** to write test cases: Put **assertion calls in test methods** to check things which are expected to be true. If they aren't, the test will fail.

- Annotations are used in place of special method name:

•Instead of using **setup** method, we can

use **@before** annotation.

•Instead of using **teardown** method, put **@after** annotation.

•Instead of using **test** before method name,

use **@test** annotation.

Q #1) What is TestNG?

A: TestNG is the framework created for executing unit tests in java program by the developers.

TestNG is an **open-source automation testing framework** based on JUnit involving more functionalities and features. So, TestNG is more powerful than Junit framework. A:18th Dec

TestNG is used to efficiently run the automated test scripts created in Selenium Webdriver.

Its full form is the “Testing New Generation” framework.

It is inspired by “JUnit” which is another framework for unit testing Java programs.

It can be integrated with Selenium or any other automation tool to provide multiple features like assertions, reporting, parallel test execution.

Que) What are the advantages of TestNG?

A The following are the advantages of TestNG are:

It generates the report in a proper format, which includes the following information:

- ✓ **Number of test cases executed.**
- ✓ **Number of test cases passed.**
- ✓ **Number of test cases failed.**
- ✓ **Number of test cases skipped**

Multiple test cases can be grouped easily by converting them into a testng.xml file, in which we can set the priority of each test case that determines which test case should be executed first.

With the help of TestNG, we can execute the multiple test cases on multiple browsers known as cross-browser testing.

- The TestNG framework can be easily integrated with other tools such as Maven, Jenkins.
- Annotations used in a TestNG framework are easily understandable such as @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest.
- WebDriver does not generate the reports while TestNG generates the reports in a readable format.

- TestNG simplifies the way the test cases are coded.
- We do not have to write the static main method.
- The sequence of actions is maintained by the annotations only.
- TestNG allows you to execute the test cases separately.

For example,

- if you have six test cases, then one method is written for each test case.
- When we run the program, five methods are executed successfully, and the sixth method is failed.