Thakur Educational Trust's (Regd.)

THAKUR RAMNARAYAN COLLEGE OF ARTS & COMMERCE

ISO 21001:2018 Certified

PROGRAMME: B.Sc (I.T)

CLASS: S.Y.B.Sc (I.T)

SUBJECT NAME: SOFTWARE ENGINEERING

SEMESTER: IV

FACULTY NAME: Ms. SMRITI DUBEY

UNIT IV

Chapter 2 – Software Testing

Concepts:

Introduction

Software Testing is a method to check whether the actual software product matches expected requirements and to **ensure that software product is Defect free**. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

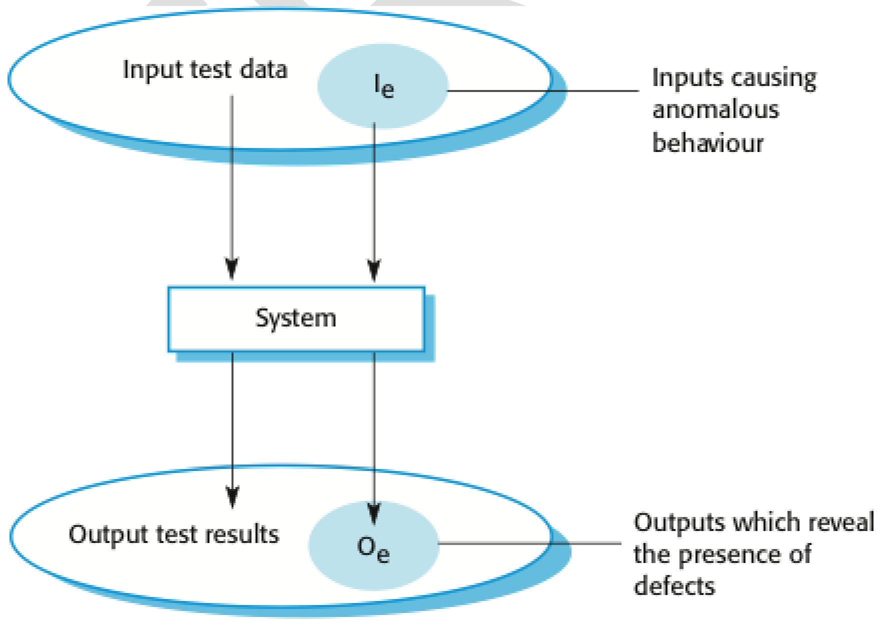
Testing is intended to **show that a program does what it is intended to do and to discover program defects before it is put into use**. When you test software, you execute a program using artificial data. You check the results of the test run for errors, anomalies or information about the program's non-functional attributes. **Testing can reveal the presence of errors, but NOT their absence.** Testing is part of a more general verification and validation process, which also includes static validation techniques.

Goals of software testing:

1. To demonstrate to the developer and the customer that the software meets its requirements.
   - Leads to validation testing: you expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
   - A successful test shows that the system operates as intended.

2. To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
   - Leads to defect testing: the test cases are designed to expose defects; the test cases can be deliberately obscure and need not reflect how the system is normally used.
   - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

Testing can be viewed as an input-output process:


Input test data
I_e
Inputs causing anomalous behaviour
System
Output test results
O_e
Outputs which reveal the presence of defects

Levels of Software Testing

Testing levels are the procedure for finding the missing areas and avoiding overlapping and repetition between the development life cycle stages. The levels of software testing involve the different methodologies, which can be used while we are performing the software testing.

In software testing, we have four different levels of testing, which are as discussed below:

1. **Unit Testing**
2. **Integration Testing**
3. **System Testing**
4. **Acceptance Testing**

Level1: Unit Testing

Unit testing is the first level of software testing, which is used to test if software modules are satisfying the given requirement or not.

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with its performance.

A unit is a single testable part of a software system and tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. White box testing approach used for unit testing and usually done by the developers.

Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or one by one, and this process is known as **Unit testing or components testing**.

Level2: Integration Testing

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.

Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as **integration testing.**

Types of Integration Testing

Integration testing can be classified into two parts:

- Incremental integration testing
- Non-incremental integration testing

Incremental Approach

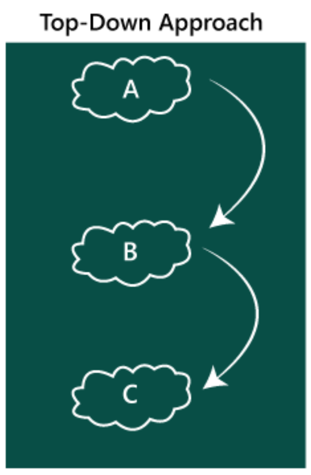
In the Incremental Approach, modules are added in ascending order one by one or according to need. The selected modules must be logically related. Generally, two or more than two modules are added and tested to determine the correctness of functions. The process continues until the successful testing of all the modules.

Incremental integration testing is carried out by further methods:

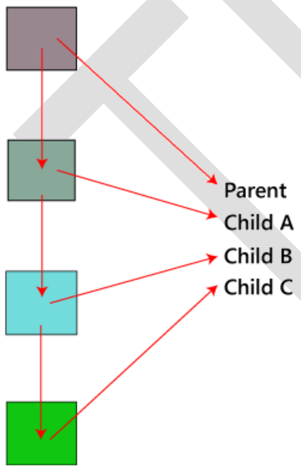
- Top-Down approach
- Bottom-Up approach

Top-Down Approach

The top-down testing strategy deals with the process in which higher level modules are tested with lower-level modules until the successful completion of testing of all the modules. Major design flaws can be detected and fixed early because critical modules tested first. In this type of method, we will add the modules incrementally or one by one and check the data flow in the same order.

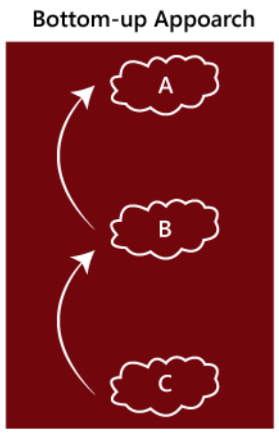


In the top-down approach, we will be ensuring that the module we are adding is the **child of the previous one like Child C is a child of Child B** and so on as we can see in the below image:

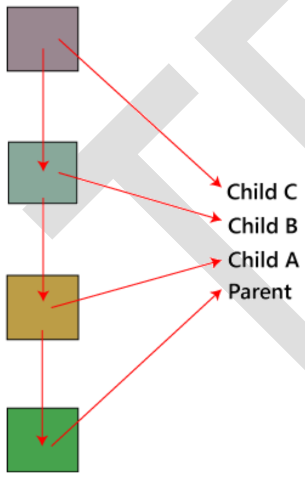


Bottom-Up Method

The bottom to up testing strategy deals with the process in which lower-level modules are tested with higher level modules until the successful completion of testing of all the modules. Top level critical modules are tested at last, so it may cause a defect. Or we can say that we will be adding the modules from bottom to the top and check the data flow in the same order.



In the bottom-up method, we will ensure that the modules we are adding **are the parent of the previous one** as we can see in the below image:



Hybrid Testing Method

In this approach, both Top-Down and Bottom-Up approaches are combined for testing. In this process, top-level modules are tested with lower-level modules and lower level modules tested with high-level modules simultaneously. There is less possibility of occurrence of defect because each module interface is tested.

Non- incremental integration testing

We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and check if the data is present. **Hence, it is also known as the Big bang method.**

Big Bang Method

In this approach, testing is done via integration of all modules at once. It is convenient for small software systems, if used for large software systems identification of defects is difficult.

Since this testing can be done after completion of all modules due to that testing team has less time for execution of this process so that internally linked interfaces and high-risk critical modules can be missed easily.

Level3: System Testing

The third level of software testing is system testing, which is used to test the software's functional and non-functional requirements.

It is end-to-end testing where the testing environment is parallel to the production environment. In the third level of software testing, we will test the application as a whole system.

To check the end-to-end flow of an application or the software as a user is known as System testing.

In system testing, we will go through all the necessary modules of an application and test if the end features or the end business works fine, and test the product as a complete system.

Level4: Acceptance Testing

The last and fourth level of software testing is acceptance testing, which is used to evaluate whether a specification or the requirements are met as per its delivery.

The software has passed through three testing levels (Unit Testing, Integration Testing, System Testing). Some minor errors can still be identified when the end-user uses the system in the actual scenario.

In simple words, we can say that Acceptance testing is the squeezing of all the testing processes that are previously done.

The acceptance testing is also known as User acceptance testing (UAT) and is done by the customer before accepting the final product.

Usually, UAT is done by the domain expert (customer) for their satisfaction and checks whether the application is working according to given business scenarios and real-time scenarios.

Component Testing

Component testing is defined as a software testing type, in which the testing is performed on each individual component separately without integrating with other components. It's also referred to as Module Testing when it is viewed from an architecture perspective. Component Testing is also referred to as Unit Testing, Program Testing or Module Testing.

Generally, any software as a whole is made of several components. Component Level Testing deals with testing these components individually.

It's one of most frequent black box testing types which is performed by QA Team.

The usage of the term "Component Testing" varies from domain to domain and organization to organization. Component testing is **performed by testers**. 'Unit Testing' is performed by the developers where they do the testing of the individual functionality or procedure. After Unit Testing is performed, the next testing is component testing. Component testing is done by the testers.

Component Testing test strategy

Depending upon the depth of testing level, component testing is divided into two parts:

1. Component Testing in Small (CTIS)
2. Component Testing in Large (CTIL)

When component testing is done in isolation with other components, it is called as component testing in small. This is done without considering integration with other components.

When component testing is done without isolation with other components of the software then it is called as component testing in large. This happens when there is a dependency on the functionality flow of the components and thus, we can't isolate them.

If the components on which we have dependency are not developed yet, then we use dummy objects in place of the actual components. These dummy objects are the stub (called function) and driver (calling function).

Release Testing

Release testing is the process of **testing a particular release** of a system that is **intended for use outside of the development team**. The primary goal of the release testing process is to **convince the customer of the system that it is good enough for use**. Release testing, therefore, has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use. Release testing is usually a black-box testing process where **tests are only derived from the system specification**.

Release testing is a form of system testing. Important differences:

- A separate team that has not been involved in the system development, should be responsible for release testing.
- System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

Requirements-based testing involves examining each requirement and developing a test or tests for it. It is validation rather than defect testing: you are trying to demonstrate that the system has properly implemented its requirements.

Scenario testing is an approach to release testing where you devise typical scenarios of use and use these to develop test cases for the system. Scenarios should be realistic and real system users should be able to relate to them. If you have used scenarios as part of the requirements engineering process, then you may be able to reuse these as testing scenarios.

Part of release testing may involve testing the **emergent properties** of a system, such as performance and reliability. Tests should reflect the profile of use of the system. Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable. Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behaviour.

Test Case design

Test case design refers to how you set-up your test cases. It is important that your tests are designed well, or you could fail to identify bugs and defects in your software during testing.

There are many different test case design techniques used to test the functionality and various features of your software. Designing good test cases ensure that every aspect of your software gets tested so that you can find and fix any issues.

A basic example of test case design:

Title: Login to the website or app
Description: User should be able to successfully log in to their account on the website/app
Preconditions: User must already be registered and use their correct login details
Assumptions: They are using a supported device or browser to log in

Test Steps:

Open website or app
Enter the username and password in the appropriate fields
Click "login"

Expected Result: The user should log in successfully.

Types of test case design techniques

The main purpose of test case design techniques is to test the functionalities and features of the software with the help of effective test cases. The test case design techniques are broadly classified into three major categories.

1. Requirement based testing
2. Partition testing
3. Structured testing

1. **Requirement based testing**: Test cases are designed to test system requirements. It is mostly used in the system design phase as the requirements are usually implemented by different components; for each requirement, test cases are identified that can prove that the system satisfies it.

2. **Partition testing**: Identify the input and output partitions and design the tests so that the system performs all the inputs of all partitions and generates all the outputs in all partitions. Partitions are groups of data that have common characteristics, such as all negative numbers, all names less than 30 characters, all events that result from the choice of objects in a menu and so on.

3. **Structured testing**: The knowledge of the program structure is used to design tests that carry out all the parts of the program. Essentially when you test a program you should try to execute each instruction at least once. The structural test helps to identify the test cases that can make this possible.

Test automation

Automation Testing is a software testing technique that performs using special automated testing software tools to execute a test case suite. On the contrary, Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps.

The automation testing software can also enter test data into the System Under Test, compare expected and actual results and generate detailed test reports. Software Test Automation demands considerable investments of money and resources.

Using a test automation tool, it’s possible to record this test suite and re-play it as required. Once the test suite is automated, no human intervention is required. This improved ROI of Test

Automation. The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate Manual Testing altogether.

Test Automation is the best way to increase the effectiveness, test coverage, and execution speed in software testing. Automated software testing is important due to the following reasons:

- Manual Testing of all workflows, all fields, all negative scenarios is time and money consuming
- It is difficult to test for multilingual sites manually
- Test Automation in software testing does not require Human intervention. You can run automated test unattended (overnight)
- Test Automation increases the speed of test execution
- Automation helps increase Test Coverage
- Manual Testing can become boring and hence error-prone.