**Unit V**
**( 07 Hrs)**

Object oriented Testing: Definition, Issues, Class Testing, Object Oriented Integration and System Testing. Testing Web Applications: What is Web testing?, User interface Testing, Usability Testing, Security Testing, Performance Testing, Database testing, Post Deployment Testing.
Linguistic –Metrics – Structural Metric – Path Products and Path Expressions. Syntax Testing – Formats – Test Cases .
Logic Based Testing – Decision Tables – Transition Testing – States, State Graph, State Testing.

**Object Oriented Testing in Software Testing**
Software typically undergoes many levels of testing, from unit testing to system or acceptance testing. Typically, in-unit testing, small "units", or modules of the software, are tested separately with focus on testing the code of that module. In higher, order testing (e.g, acceptance testing), the entire system (or a subsystem) is tested with the focus on testing the functionality or external behavior of the system.

As information systems are becoming more complex, the object-oriented paradigm is gaining popularity because of its benefits in analysis, design, and coding. Conventional testing methods cannot be applied for testing classes because of problems involved in testing classes, abstract classes, inheritance, dynamic binding, message, passing, polymorphism, concurrency, etc.
Testing classes is a fundamentally different problem than testing functions. A function (or a procedure) has a clearly defined input-output behavior, while a class does not have an input-output behavior specification. We can test a method of a class using approaches for testing functions, but we cannot test the class using these
approaches.

According to Davis the dependencies occurring in conventional systems are:

Data dependencies between variables
Calling dependencies between modules
Functional dependencies between a module and the variable it computes
Definitional dependencies between a variable and its types.

But in Object-Oriented systems there are following additional dependencies:

Class to class dependencies
Class to method dependencies
Class to message dependencies

Class to variable dependencies
Method to variable dependencies
Method to message dependencies
Method to method dependencies

**Issues in Testing Classes:**
Additional testing techniques are, therefore, required to test these dependencies. Another issue of interest is that it is not possible to test the class dynamically, only its instances i.e, objects can be tested. Similarly, the concept of inheritance opens various issues e.g., if changes are made to a parent class or superclass, in a larger system of a class it will be difficult to test subclasses individually and isolate the error to one class.

In object-oriented programs, control flow is characterized by message passing among objects, and the control flow switches from one object to another by inter-object communication. Consequently, there is no control flow within a class like functions. This lack of sequential control flow within a class requires different approaches for testing. Furthermore, in a function, arguments passed to the function with global data determine the path of execution within the procedure. But, in an object, the state associated with the object also influences the path of execution, and methods of a class can communicate among themselves through this state because this state is persistent across invocations of methods. Hence, for testing objects, the state of an object has to play an important role.

Techniques of object-oriented testing are as follows:

1. **Fault Based Testing:**
   This type of checking permits for coming up with test cases supported the consumer specification or the code or both. It tries to identify possible faults (areas of design or code that may lead to errors.). For all of these faults, a test case is developed to “flush” the errors out. These tests also force each time of code to be executed.

This method of testing does not find all types of errors. However, incorrect specification and interface errors can be missed. These types of errors can be uncovered by function testing in the traditional testing model. In the object-oriented model, interaction errors can be uncovered by scenario-based testing. This form of Object oriented-testing can only test against the client’s specifications, so interface errors are still missed.

2. 
3. **Class Testing Based on Method Testing:**
   This approach is the simplest approach to test classes. Each method

of the class performs a well defined cohesive function and can, therefore, be related to unit testing of the traditional testing techniques. Therefore all the methods of a class can be involved at least once to test the class.

4. **Random Testing:**
   It is supported by developing a random test sequence that tries the minimum variety of operations typical to the behavior of the categories
5. **Partition Testing:**
   This methodology categorizes the inputs and outputs of a category so as to check them severely. This minimizes the number of cases that have to be designed.
6. **Scenario-based Testing:**
   It primarily involves capturing the user actions then stimulating them to similar actions throughout the test.
   These tests tend to search out interaction form of error.

## Object Oriented Integration and System Testing

### Testing Object-Oriented Systems

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, Integration testing, and system testing.

**Unit Testing:** In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free. Unit testing is the responsibility of the application engineer who implements the structure.

**Integration Testing:** This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside. Subsystem tests can be used as regression tests for each newly released version of the subsystem.

**System Testing:** System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.

### Object-Oriented Testing Techniques

### Grey Box Testing

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are −

**State model based testing** − This encompasses state coverage, state transition coverage, and state transition path coverage.

**Use case based testing** − Each scenario in each use case is tested.

**Class diagram based testing** − Each class, derived class, associations, and aggregations are tested.

**Sequence diagram based testing** − The methods in the messages in the sequence diagrams are tested.

**Techniques for Integration Testing**
The two main approaches of Integration testing are −

**Thread based testing** − All classes that are needed to realize a single use case in a subsystem are integrated and tested.

**Use based testing** − The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

**Categories of System Testing**

**Alpha testing** − This is carried out by the testing team within the organization that develops software.

**Beta testing** − This is carried out by select group of co-operating customers.

**Acceptance testing** − This is carried out by the customer before accepting the deliverables.

# What is Web Testing? Types of Web Application Testing

Web testing, at its core, is simply checking your web application or your website for problems before you make that web application or website live. Web testing is designed to check all aspects of the web application's functionality, including looking for bugs with usability, compatibility, security, and general performance.

Web testing is a crucial part of assembling any web application or website, as you don't want to invest the many resources in time and money you've spent developing this web application and then have it run into immediate problems upon release. We have seen that happen before, and it isn't pretty.

## What Are the Types of Web Application Testing?

In an era where there's so much software in a computer, improper testing of a computer program can lead to catastrophic results that cost millions – even billions – to fix. Proper use of a web testing application, or web testing automation, can be used in the following ways:

## 1. Testing the Functionality and Features

Functionality testing is virtually the most basic, yet extremely crucial for any application – including web. Functionality testing ensures a web application is working properly and correctly. Web testing tools will inspect factors like making sure every link on the website points to the right page.

Functional testing covers:

> **Unit Testing:** This stage of functional testing validates small and individual areas of the application in the early stages of development to reduce the chance of escalating to more severe bugs later on
>
> **Smoke Testing, Build Verification Testing and Confidence Testing:** After each build, this test is run to verify the web application is stable and ready for further testing to avoid wasting testing efforts
>
> **Sanity Testing:** Once the build verification is complete, this test checks the new code introduced and specific functionalities
>
> **Regression Testing:** Retest a selective list of test cases to identify areas that reacts more severe to changes and ensure existing features stay functional
>
> **Integration Testing:** locate faults on the linkage of interconnected modules (E.g., being redirected to a Mailbox page after successfully signing up)

- **Usability Testing:** Used to find areas for improvement to the overall UX design according to a real user's behavior and feedback

Functional testing also tests forms to make sure that they're working.

## 2. Testing Web APIs

Web APIs, as the name suggests, are application programming interfaces for the web. Testing the API requires making requests to multiple API endpoints to validate the response. This can measure many things, including function, security, and performance. API testing is essential because it tests the logic, responses, security, and performance bottlenecks.

For example, say you want to test a login form. Particularly, you want every username and password entered to get properly stored in the database.

In API terms, you're looking at the transfer of data between end-points. To encrypt and shield the HTTP data exchanges from potential hackers, work with negative test cases like:

- Accessing the APIs without following the authentication rule.
- Test invalid values in JSONS (e.g., Cannot register user with a non-existing email address)

*>>> Learn more about API testing and the most popular tools:* Best Automated API Testing Tools for Software Testing in 2022

Cookie testing is another facet to look into when sending user-specific requests and sessions. With APIs as the middleman, parameters to delete or store information from users' sessions and activities are often added to API functional test cases.

## 3. Testing the Database

Database testing makes sure that the data values and information stored in the database are valid. It will help to prevent data loss, save lost transaction data, and prevent unauthorized access to the information.

The testing involves checking the schema, tables, and triggers of common databases such as  Excel/CSV, GraphQL, Oracle SQL and SQL Server. It often includes stress testing and using complex queries on a single or a combination of data files. Database testing is the confidence provider in assuring all data transmissions are successfully done, regardless of the pressure put on.

## What is User Interface Testing?

User interface testing, a testing technique used to identify the presence of defects is a product/software under test by using Graphical user interface [GUI].

## GUI Testing - Characteristics:

GUI is a hierarchical, graphical front end to the application, contains graphical objects with a set of properties.

During execution, the values of the properties of each objects of a GUI define the GUI state.

It has capabilities to exercise GUI events like key press/mouse click.

Able to provide inputs to the GUI Objects.

To check the GUI representations to see if they are consistent with the expected ones.

It strongly depends on the used technology.

## GUI Testing - Approaches:

**Manual Based -** Based on the domain and application knowledge of the tester.

**Capture and Replay -** Based on capture and replay of user actions.

**Model-based testing -** Based on the execution of user sessions based on a GUI model. Various GUI models are briefly discussed below.

## Model Based Testing - In Brief:

**Event-based model -** Based on all events of the GUI need to be executed at least once.

**State-based model** - "all states" of the GUI are to be exercised at least once.

**Domain model -** Based on the application domain and its functionality.

## GUI Testing Checklist:

Check Screen Validations

Verify All Navigations

Check usability Conditions

Verify Data Integrity

Verify the object states

Verify the date Field and Numeric Field Formats

## GUI Automation Tools

Following are some of the open source GUI automation tools in the market:

| Product | Licensed Under | URL |
|---|---|---|
| AutoHotkey | GPL | http://www.autohotkey.com/ |
| Selenium | Apache | http://docs.seleniumhq.org/ |
| Sikuli | MIT | http://sikuli.org |
| Robot Framework | Apache | www.robotframework.org |
| watir | BSD | http://www.watir.com/ |
| Dojo Toolkit | BSD | http://dojotoolkit.org/ |

Following are some of the Commercial GUI automation tools in the market.

| Product | Vendor | URL |
|---|---|---|
| AutoIT | AutoIT | http://www.autoitscript.com/site/autoit/ |
| EggPlant | TestPlant | www.testplant.com |
| QTP | Hp | http://www8.hp.com/us/en/software-solutions/ |

| | | |
|---|---|---|
| Rational Functional Tester | IBM | http://www-03.ibm.com/software/products/us/en/functional |
| Infragistics | Infragistics | www.infragistics.com |
| iMacros | iOpus | http://www.iopus.com/iMacros/ |
| CodedUI | Microsoft | http://www.microsoft.com/visualstudio/ |
| Sikuli | Micro Focus International | http://www.microfocus.com/ |

## What is usability testing?

Usability testing is a type of user research that evaluates the user's experience when interacting with a website or app. It helps your designers and product teams assess how intuitive and easy-to-use products are.
The usability testing process identifies problems with your product you may have otherwise missed—by asking real users (instead of developers or designers) to complete a series of usability tasks on the product. The results, success rate, and paths taken to complete the tasks are then analyzed to identify potential issues and areas for improvement.
The ultimate goal of usability testing is to create a product that solves your user's problems and helps them achieve their objectives with a positive experience.

### What usability testing is not

Before we can delve into the specifics of usability testing, we need to dispel some misconceptions. User research, user testing, and usability testing are often used interchangeably—but they aren't necessarily the same.

User research refers to the overarching act of gathering feedback and insights from product users, and using this to inform product decisions. **Usability testing is a specific *kind* of user research done to assess the usability of a product or design.**
To help clarify, here are some UX research methods and techniques that *don't* qualify as usability testing:

**A/B testing:** A/B testing is a statistical method of comparing and testing design variations to identify which the user prefers, and which changes improve performance. A/B testing is about preferences and provides only quantitative data, whereas usability testing is about behavior, and can provide both quantitative and qualitative data.

**Surveys**: Surveys can help you collect data from a large number of users or customers, typically in the form of multiple-choice questions, sentiment, or ratings. Surveys are great for understanding how users feel about a product, but don't show you how they interact with it.

**Focus groups:** During a focus group, a moderator guides the discussion and asks open-ended questions to encourage participants to share their opinions and ideas. The goal is to gain insights into people's attitudes and experiences related to the topic.

**User testing**: This is a broad term that can either refer to user research as a whole, or specifically the process of testing products and ideas with real users. The latter takes a quantitative approach to collect user feedback, and typically comes before usability testing. It doesn't give you qualitative data about why users struggle to complete tasks.

**Security Testing** is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss. Security testing of any system is focused on finding all possible loopholes and weaknesses of the system which might result in the loss of information or repute of the organization. Security testing is a type of software testing that focuses on evaluating the security of a system or application. The goal of security testing is to identify vulnerabilities and potential threats, and to ensure that the system is protected against unauthorized access, data breaches, and other security-related issues.

**Goal of Security Testing:** The goal of security testing is to:

To identify the threats in the system.
To measure the potential vulnerabilities of the system.
To help in detecting every possible security risks in the system.
To help developers in fixing the security problems through coding.

The goal of security testing is to identify vulnerabilities and potential threats in a system or application, and to ensure that the system is protected against unauthorized access, data breaches, and other security-related issues. The main objectives of security testing are to:

Identify vulnerabilities: Security testing helps identify vulnerabilities in the system, such as weak passwords, unpatched software, and misconfigured systems, that could be exploited by attackers.

Evaluate the system's ability to withstand an attack: Security testing evaluates the system's ability to withstand different types of attacks, such as network attacks, social engineering attacks, and application-level attacks.

Ensure compliance: Security testing helps ensure that the system meets relevant security standards and regulations, such as HIPAA, PCI DSS, and SOC2.

Provide a comprehensive security assessment: Security testing provides a comprehensive assessment of the system's security posture, including the identification of vulnerabilities, the evaluation of the system's ability to withstand an attack, and compliance with relevant security standards.

Help organizations prepare for potential security incidents: Security testing helps organizations understand the potential risks and vulnerabilities that they face, enabling them to prepare for and respond to potential security incidents.

Identify and fix potential security issues before deployment to production: Security testing helps identify and fix security issues before the system is deployed to production. This helps reduce the risk of a security incident occurring in a production environment.

**Principle of Security Testing:** Below are the six basic principles of security testing:

Confidentiality
Integrity
Authentication
Authorization
Availability
Non-repudiation

**Major Focus Areas in Security Testing:**

Network Security
System Software Security
Client-side Application Security
Server-side Application Security

Authentication and Authorization: Testing the system's ability to properly authenticate and authorize users and devices. This includes testing the strength and effectiveness of passwords, usernames, and other forms of authentication, as well as testing the system's access controls and permission mechanisms.

Network and Infrastructure Security: Testing the security of the system's network and infrastructure, including firewalls, routers, and other network devices. This includes testing the system's ability to defend against common network attacks such as denial of service (DoS) and man-in-the-middle (MitM) attacks.

Database Security: Testing the security of the system's databases, including testing for SQL injection, cross-site scripting, and other types of attacks.

Application Security: Testing the security of the system's applications, including testing for cross-site scripting, injection attacks, and other types of vulnerabilities.

Data Security: Testing the security of the system's data, including testing for data encryption, data integrity, and data leakage.

Compliance: Testing the system's compliance with relevant security standards and regulations, such as HIPAA, PCI DSS, and SOC2.

Cloud Security: Testing the security of cloud-

**Types of Security Testing:**

1. **Vulnerability Scanning:** Vulnerability scanning is performed with the help of automated software to scan a system to detect the known vulnerability patterns.
2. **Security Scanning:** Security scanning is the identification of network and system weaknesses. Later on it provides solutions for

reducing these defects or risks. Security scanning can be carried out in both manual and automated ways.

3. **Penetration Testing:** Penetration testing is the simulation of the attack from a malicious hacker. It includes an analysis of a particular system to examine for potential vulnerabilities from a malicious hacker that attempts to hack the system.
4. **Risk Assessment:** In risk assessment testing security risks observed in the organization are analyzed. Risks are classified into three categories i.e., low, medium and high. This testing endorses controls and measures to minimize the risk.
5. **Security Auditing:** Security auditing is an internal inspection of applications and operating systems for security defects. An audit can also be carried out via line-by-line checking of code.
6. **Ethical Hacking:** Ethical hacking is different from malicious hacking. The purpose of ethical hacking is to expose security flaws in the organization's system.
7. **Posture Assessment:** It combines security scanning, ethical hacking and risk assessments to provide an overall security posture of an
8. **Application security testing:** Application security testing is a type of testing that focuses on identifying vulnerabilities in the application itself. It includes testing the application's code, configuration, and dependencies to identify any potential vulnerabilities.
9. **Network security testing**: Network security testing is a type of testing that focuses on identifying vulnerabilities in the network infrastructure. It includes testing firewalls, routers, and other network devices to identify potential vulnerabilities.
10. **Social engineering testing**: Social engineering testing is a type of testing that simulates phishing, baiting, and other types of social engineering attacks to identify vulnerabilities in the system's human element.
11. **Tools such as Nessus,** OpenVAS, and Metasploit can be used to automate and simplify the process of security testing. It's important to ensure that security testing is done regularly and that any vulnerabilities or threats identified during testing are fixed immediately to protect the system from potential attacks. organization.

**Performance Testing** is a type of software testing that ensures software applications to perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity and stability under a particular workload.

Performance testing is a type of software testing that focuses on evaluating the performance and scalability of a system or application. The goal of performance testing is to identify bottlenecks, measure system performance under various loads and conditions, and ensure that the system can handle the expected number of users or transactions.

## There are several types of performance testing, including:

**Load testing:** Load testing simulates a real-world load on the system to see how it performs under stress. It helps identify bottlenecks and determine the maximum number of users or transactions the system can handle.

**Stress testing:** Stress testing is a type of load testing that tests the system's ability to handle a high load above normal usage levels. It helps identify the breaking point of the system and any potential issues that may occur under heavy load conditions.

**Spike testing:** Spike testing is a type of load testing that tests the system's ability to handle sudden spikes in traffic. It helps identify any issues that may occur when the system is suddenly hit with a high number of requests.

**Soak testing:** Soak testing is a type of load testing that tests the system's ability to handle a sustained load over a prolonged period of time. It helps identify any issues that may occur after prolonged usage of the system.

**Endurance testing:** This type of testing is similar to soak testing, but it focuses on the long-term behavior of the system under a constant load.

Performance Testing is the process of analyzing the quality and capability of a product. It is a testing method performed to determine the system performance in terms of speed, reliability and stability under varying workload. Performance testing is also known as ***Perf Testing***.

**Performance Testing Attributes:**

**Speed:**

It determines whether the software product responds rapidly.

**Scalability:**

It determines amount of load the software product can handle at a time.

**Stability:**

It determines whether the software product is stable in case of varying workloads.

**Reliability:**

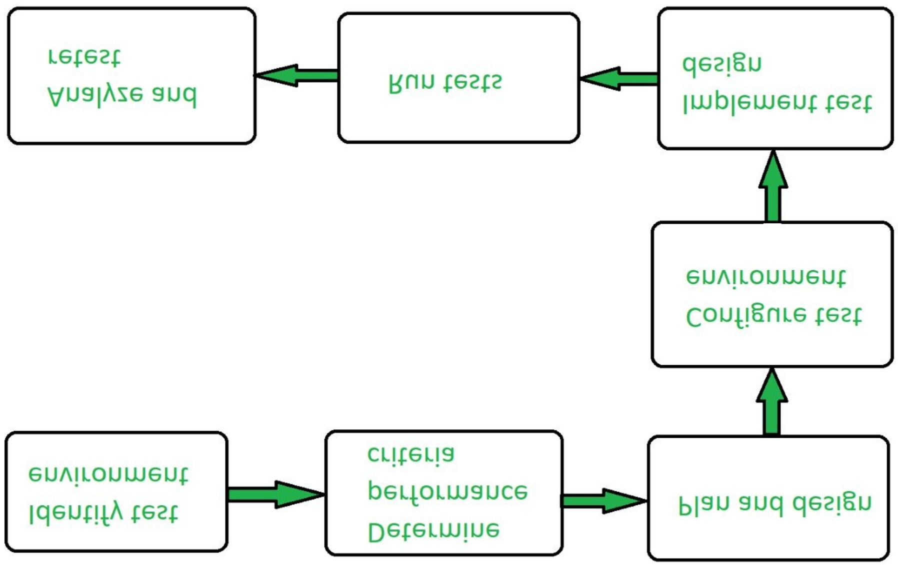
It determines whether the software product is secure or not.

1. The objective of performance testing is to eliminate performance congestion.
2. It uncovers what is needed to be improved before the product is launched in market.
3. The objective of performance testing is to make software rapid.
4. The objective of performance testing is to make software stable and reliable.
5. The objective of performance testing is to evaluate the performance and scalability of a system or application under various loads and conditions. It helps identify bottlenecks, measure system performance, and ensure that the system can handle the expected number of users or transactions. It also helps to ensure that the system is reliable, stable and can handle the expected load in a production environment.

**Types of Performance Testing:**

1. ***Load testing***:
   It checks the product's ability to perform under anticipated user loads. The objective is to identify performance congestion before the software product is launched in market.
2. ***Stress testing***:
   It involves testing a product under extreme workloads to see whether it handles high traffic or not. The objective is to identify the breaking point of a software product.
3. ***Endurance testing***:
   It is performed to ensure the software can handle the expected load over a long period of time.
4. ***Spike testing***:
   It tests the product's reaction to sudden large spikes in the load generated by users.
5. ***Volume testing***:
   In volume testing large number of data is saved in a database and the overall software system's behavior is observed. The objective is to check product's performance under varying database volumes.
6. ***Scalability testing***:
   In scalability testing, software application's effectiveness is determined in scaling up to support an increase in user load. It helps in planning capacity addition to your software system.

**Performance Testing Process:**


Identify test environment
Determine performance criteria
Plan and design
Configure test environment
Implement test design
Run tests
Analyze and retest

Performance Testing Tools:

1. Jmeter
2. Open STA
3. Load Runner
4. Web Load

## What is Database Testing?

**Database Testing** is a type of software testing that checks the schema, tables, triggers, etc. of the Database under test. It also checks data integrity and consistency. It may involve creating complex queries to load/stress test the Database and check its responsiveness.

**Database Testing is important** in software testing because it ensures data values and information received and stored into database are valid or not. Database testing helps to save data loss, saves aborted transaction data and no unauthorized access to the information. Database is important for any software application hence testers must have good knowledge of SQL for database testing.

The GUI is usually given the most emphasis by the test and development team members since the Graphical User Interface happens to be the most visible part of the application.

However, what is also important is to validate the information that is the heart of the application, aka DATABASE.

Let us consider a Banking application wherein a user makes transactions. Now from Database Testing or DB Testing viewpoint following, things are important:

1. The application stores the transaction information in the application database and displays them correctly to the user.
2. No information is lost in the process.
3. No partially performed or aborted operation information is saved by the application.
4. No unauthorized individual is allowed to access the user's information.

To ensure all these above objectives, we need to use data validation or data testing.

## Differences between User-Interface Testing and Data Testing

| User-Interface testing | Database or Data testing |
|---|---|
| This type of testing is also known as Graphical User Interface testing or Front-end Testing. | This type of testing is also kno Backend Testing or data testin |
| This type of testing chiefly deals with all the testable items that are open to the user for viewership and interaction like Forms, Presentation, Graphs, Menus, and Reports, etc. (created through VB, VB.net, VC++, Delphi – Front-end Tools) | This type of testing chiefly dea the testable items that are gener from the user for viewership. Th internal processes and storage Assembly, DBMS like Oracle, S MYSQL, etc. |
| **This type of testing includes validating the** | **This type of testing involves va** |

| User-Interface testing | Database or Data testing |
|---|---|
| text boxes<br>select dropdowns<br>calendars and buttons<br>Page navigation<br>display of images<br>Look and feel of the overall application | the schema<br>database tables<br>columns<br>keys and indexes<br>stored procedures triggers<br>database server validation<br>validating data duplicatio |
| The tester must be thoroughly knowledgeable about the business requirements as well as the usage of the development tools and the usage of automation frameworks and tools. | To be able to perform backend must the tester have a strong b in the database server and Stru Query Language concepts. |

**Post-deployment** testing is a type of testing in which the software is tested after it is being deployed to production. This testing may help us in finding those problems which were not detected before its deployment in the production and despite all the planning and testing carried out before the final deployment, obtaining user opinion is important for the improvement of a website. It ensures that the website adapts to the needs of the user. User feedback may come in various forms, ranging from reporting faults to suggestions for improvement of the website.

## Need for Post Deployment Testing

What do we need to test the application post its deployment and how? There are various ways for getting the user's feedback about the software application but the effective way to know the opinion is by getting a survey form filled by every user because these survey forms can be used to detect the latest trends from the users and also provide some good information about the software improvement. The response obtained from these surveys may help the tester or developer some improvement in their app.

Now after receiving the user feedback, the testers need to identify the useful fault reporting, suggestions, and recommendations that the users have given. For these given things the following criteria can be used for checking which feedback is more important:

1. Calculating the total users that have given the same feedback or recommendation. If the feedback is not very popular among the group of users so, then we must think twice before implementing their suggestion in the software.
2. Are the users fake or Real? It is vital to make sure that the Source of feedback: suggestions come from regular users and not accidental users.
3. Is the suggested idea worth implementing? Before implementing the suggested changes, the cost and schedule must be analyzed carefully and there should be a thorough check of the correctness of the proposed change and its impact on the application. And the benefits of implementing the suggested changes in the software must be determined by the testers or developers.
4. Will the feedback or recommendation increase its impact on the software? Is the recommendation compatible with other features also? Getting this feedback is very significant and answering them is very crucial and making the results of implementing these recommendations are sometimes unpredictable.

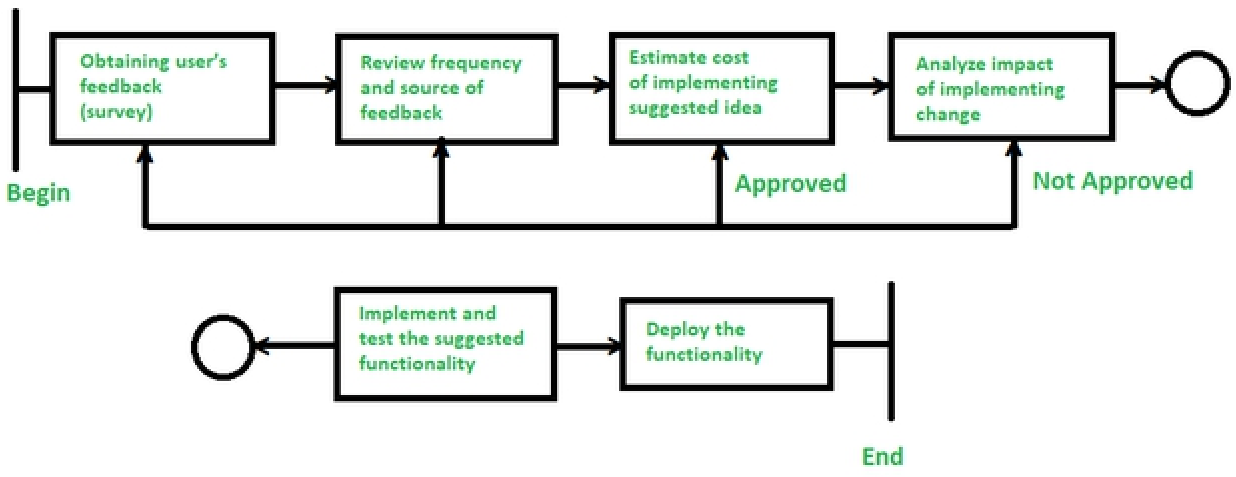
## Post Deployment Testing Activities

**Post-deployment verification:** Firstly, the QA or Test lead conducts the verification of the software application as per the need and its requirement and these are generally the test plans and the test cases of the software application which are compared from the previous or existing test plan and test cases.

**Reporting the post-deployment verification results:** After verifying the above cases, the results with all the necessary information are then told to the clients and if they find any issues/defects in the production then the issues are told to the test lead, and further new testing will be conducted.

**Removing the test data:** Once the results are positive, the test data that was used in the post-deployment testing are then removed from the process.

**Monitoring the application: Finally**, if there's any need for monitoring the application in the production then that is also done.


Begin
Obtaining user's feedback (survey)
Review frequency and source of feedback
Estimate cost of implementing suggested idea
Analyze impact of implementing change
Approved
Not Approved
Implement and test the suggested functionality
Deploy the functionality
End

*A general post-deployment testing procedure of an application*

The above procedure is important for the client to ensure that their suggested functionalities of the software are properly addressed, fixed, and removed otherwise the client may have to invest a lot of cost and effort for testing the application again.

Structural Metrics—Metrics based on structural relations between objects in the program—usually metrics on properties of control flowgraphs or data flowgraphs; for example, number of links, number of nodes, nesting depth. Linguistic metrics measure some property of text without interpreting what is measured.

## Software Testing Metrics, its Types and Example

Software testing metrics are quantifiable indicators of the software testing process progress, quality, productivity, and overall health. The purpose of software testing metrics is to increase the efficiency and effectiveness of the software testing process while also assisting in making better decisions for future testing by providing accurate data about the testing process. A metric expresses the degree to which a system, system component, or process possesses a certain attribute in numerical terms. A weekly mileage of an automobile compared to its ideal mileage specified by the manufacturer is an excellent illustration of metrics. Here, we discuss the following points:

1. **Importance of Metrics in Software Testing.**
2. **Types of Software Testing Metrics.**

3. **Manual Test Metrics: What Are They and How Do They Work?**
4. **Other Important Metrics.**
5. **Test Metrics Life Cycle.**
6. **Formula for Test Metrics.**
7. **Example of Software Test Metrics Calculation.**

## Importance of Metrics in Software Testing

Test metrics are essential in determining the software's quality and performance. Developers may use the right software testing metrics to improve their productivity.

Test metrics help to determine what types of enhancements are required in order to create a defect-free, high-quality software product.

Make informed judgments about the testing phases that follow, such as project schedule and cost estimates.

Examine the current technology or procedure to see if it need any more changes.

## Types of Software Testing Metrics

Software testing metrics are divided into three categories:

1. **Process Metrics:** A project's characteristics and execution are defined by process metrics. These features are critical to the SDLC process's improvement and maintenance (Software Development Life Cycle).
2. **Product Metrics:** A product's size, design, performance, quality, and complexity are defined by product metrics. Developers can improve the quality of their software development by utilizing these features.
3. **Project Metrics:** Project Metrics are used to assess a project's overall quality. It is used to estimate a project's resources and deliverables, as well as to determine costs, productivity, and flaws.

It is critical to determine the appropriate testing metrics for the process. A few points to keep in mind:

Before creating the metrics, carefully select your target audiences.
Define the aim for which the metrics were created.
Prepare measurements based on the project's specific requirements. Assess the financial gain associated with each statistic.

- Match the measurements to the project lifestyle phase for the best results.

The major benefit of automated testing is that it allows testers to complete more tests in less time while also covering a large number of variations that would be practically difficult to calculate manually.

## Manual Test Metrics: What Are They and How Do They Work?

Manual testing is carried out in a step-by-step manner by quality assurance experts. Test automation frameworks, tools, and software are used to execute tests in automated testing. There are advantages and disadvantages to both human and automated testing. Manual testing is a time-consuming technique, but it allows testers to deal with more complicated circumstances. There are two sorts of manual test metrics:

**1. Base Metrics:** Analysts collect data throughout the development and execution of test cases to provide base metrics. By generating a project status report, these metrics are sent to test leads and project managers. It is quantified using calculated metrics.

- The total number of test cases
- The total number of test cases completed.

**2. Calculated Metrics:** Data from base metrics are used to create calculated metrics. The test lead collects this information and transforms it into more useful information for tracking project progress at the module, tester, and other levels. It's an important aspect of the SDLC since it allows developers to make critical software changes.

## Other Important Metrics

The following are some of the other important software metrics:

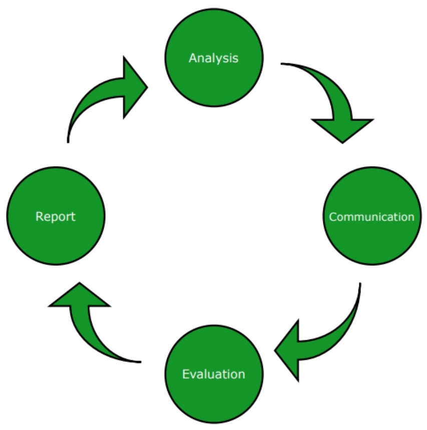
- **Defect metrics:** Defect metrics help engineers understand the many aspects of software quality, such as functionality, performance, installation stability, usability, compatibility, and so on.
- **Schedule Adherence:** Schedule Adherence's major purpose is to determine the time difference between a schedule's expected and actual execution times.
- **Defect Severity:** The severity of the problem allows the developer to see how the defect will affect the software's quality.
- **Test case efficiency:** Test case efficiency is a measure of how effective test cases are at detecting problems.
- **Defects finding rate:** It is used to determine the pattern of flaws over a period of time.
- **Defect Fixing Time:** The amount of time it takes to remedy a problem is known as defect fixing time.

**Test Coverage:** It specifies the number of test cases assigned to the program. This metric ensures that the testing is completed completely. It also aids in the verification of code flow and the testing of functionality.

**Defect cause:** It's utilized to figure out what's causing the problem.

## Test Metrics Life Cycle

The below diagram illustrates the different stages in the test metrics life cycle.



*Test Metrics Lifecycle*

The various stages of the test metrics lifecycle are:

1. **Analysis:**
   The metrics must be recognized.
   Define the QA metrics that have been identified.
2. **Communicate:**
   Stakeholders and the testing team should be informed about the requirement for metrics.
   Educate the testing team on the data points that must be collected in order to process the metrics.
3. **Evaluation:**
   Data should be captured and verified.
   Using the data collected to calculate the value of the metrics
4. **Report:**
   Create a strong conclusion for the paper.
   Distribute the report to the appropriate stakeholder and representatives.
   Gather input from stakeholder representatives.

## Formula for Test Metrics

To get the percentage execution status of the test cases, the following formula can be used:

> *Percentage test cases executed = (No of test cases executed / Total no of test cases written) x 100*

Similarly, it is possible to calculate for other parameters also such as test cases that were not executed, test cases that were passed, test cases that were failed, test cases that were blocked, and so on. Below are some of the formulas:

**1. Test Case Effectiveness:**

> *Test Case Effectiveness = (Number of defects detected / Number of test cases run) x 100*

**2. Passed Test Cases Percentage:** Test Cases that Passed Coverage is a metric that indicates the percentage of test cases that pass.

> *Passed Test Cases Percentage = (Total number of tests ran / Total number of tests executed) x 100*

**3. Failed Test Cases Percentage:** This metric measures the proportion of all failed test cases.

> *Failed Test Cases Percentage = (Total number of failed test cases / Total number of tests executed) x 100*

**4. Blocked Test Cases Percentage:** During the software testing process, this parameter determines the percentage of test cases that are blocked.

> *Blocked Test Cases Percentage = (Total number of blocked tests / Total number of tests executed) x 100*

**5. Fixed Defects Percentage:** Using this measure, the team may determine the percentage of defects that have been fixed.

> *Fixed Defects Percentage = (Total number of flaws fixed / Number of defects reported) x 100*

**6. Rework Effort Ratio:** This measure helps to determine the rework effort ratio.

> *Rework Effort Ratio = (Actual rework efforts spent in that phase/ Total actual efforts spent in that phase) x 100*

**7. Accepted Defects Percentage:** This measures the percentage of defects that are accepted out of the total accepted defects.

*Accepted Defects Percentage = (Defects Accepted as Valid by Dev Team / Total Defects Reported) x 100*

**8. Defects Deferred Percentage:** This measures the percentage of the defects that are deferred for future release.

*Defects Deferred Percentage = (Defects deferred for future releases / Total Defects Reported) x 100*

## Example of Software Test Metrics Calculation

Let's take an example to calculate test metrics:

| S No. | Testing Metric | Data retrieved during test case development |
|---|---|---|
| 1 | No. of requirements | 5 |
| 2 | The average number of test cases written per requirement | 40 |
| 3 | Total no. of Test cases written for all requirements | 200 |
| 4 | Total no. of Test cases executed | 164 |
| 5 | No. of Test cases passed | 100 |
| 6 | No. of Test cases failed | 60 |
| 7 | No. of Test cases blocked | 4 |
| 8 | No. of Test cases unexecuted | 36 |
| 9 | Total no. of defects identified | 20 |
| 10 | Defects accepted as valid by the | 15 |

| S No. | Testing Metric | Data retrieved during test case development |
|---|---|---|
| | dev team | |
| 11 | Defects deferred for future releases | 5 |
| 12 | Defects fixed | 12 |

***1. Percentage test cases executed =** (No of test cases executed / Total no of test cases written) x 100*
*= (164 / 200) x 100*
*= 82*

***2. Test Case Effectiveness =** (Number of defects detected / Number of test cases run) x 100*
*= (20 / 164) x 100*
*= 12.2*

***3. Failed Test Cases Percentage =** (Total number of failed test cases / Total number of tests executed) x 100*
*= (60 / 164) * 100*
*= 36.59*

***4. Blocked Test Cases Percentage =** (Total number of blocked tests / Total number of tests executed) x 100*
*= (4 / 164) * 100*
*= 2.44*

***5. Fixed Defects Percentage =** (Total number of flaws fixed / Number of defects reported) x 100*
*= (12 / 20) * 100*
*= 60*

***6. Accepted Defects Percentage =** (Defects Accepted as Valid by Dev Team / Total Defects Reported) x 100*
*= (15 / 20) **

*100*

*= 75*

***7. Defects Deferred Percentage =** (Defects deferred for future releases / Total Defects Reported) x 100*

*= (5 / 20) * 100*

*= 25*

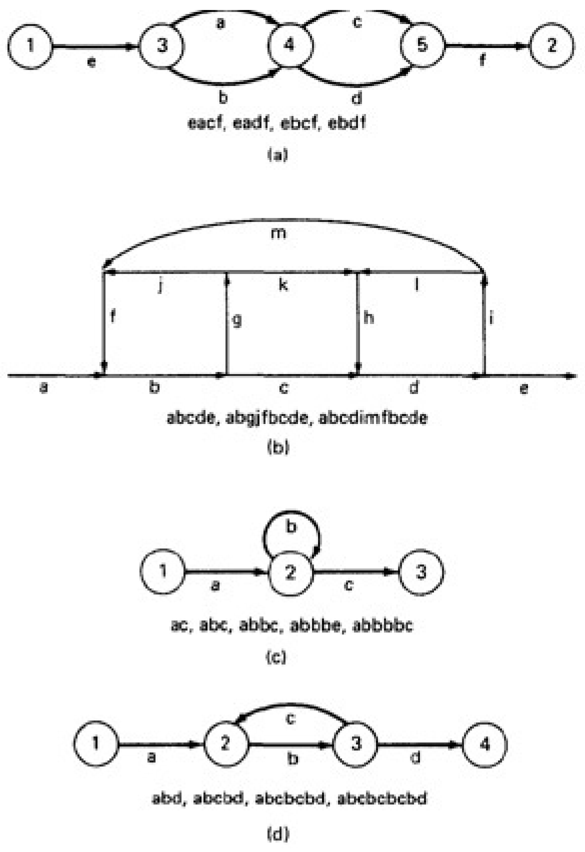
**PATHS, PATH PRODUCTS AND REGULAR EXPRESSIONS**

**MOTIVATION:**

- Flow graphs are being an abstract representation of programs.
- Any question about a program can be cast into an equivalent question about an appropriate flowgraph.
- Most software development, testing and debugging tools use flow graphs analysis techniques.

**PATH PRODUCTS:**

- Normally flow graphs used to denote only control flow connectivity.
- The simplest weight we can give to a link is a name.
- Using link names as weights, we then convert the graphical flow graph into an equivalent algebraic like expressions which denotes the set of all possible paths from entry to exit for the flow graph.
- Every link of a graph can be given a name.
- The link name will be denoted by lower case italic letters.
- In tracing a path or path segment through a flow graph, you traverse a succession of link names.
- The name of the path or path segment that corresponds to those links is expressed naturally by concatenating those link names.
    - For example, if you traverse links a,b,c and d along some path, the name for that path segment is abcd. This path name is also called a **path product.** Figure 5.1 shows some examples:


a
c
1
e
3
4
5
f
2
b
d
eacf, eadf, ebcf, ebdf
(a)
m
j
k
l
f
g
h
i
a
b
c
d
e
abcde, abgjfbcde, abcdimfbcde
(b)
b
1
a
2
c
3
ac, abc, abbc, abbbe, abbbbc
(c)
c
1
a
2
b
3
d
4
abd, abcbd, abcbcbd, abcbcbcbd
(d)

o

*Figure 5.1: Examples of paths.*

o

**PATH EXPRESSION:**

o Consider a pair of nodes in a graph and the set of paths between those node.

o Denote that set of paths by Upper case letter such as X,Y. From Figure 5.1c, the members of the path set can be listed as follows:

ac, abc, abbc, abbbc, abbbbc.............

o

o Alternatively, the same set of paths can be denoted by :

ac+abc+abbc+abbbc+abbbbc+...........

o

o The + sign is understood to mean "or" between the two nodes of interest, paths ac, or abc, or abbc, and so on can be taken.

o Any expression that consists of path names and "OR"s and which denotes a set of paths between two nodes is called a "**Path Expression.**".

**PATH PRODUCTS:**

- The name of a path that consists of two successive path segments is conveniently expressed by the concatenation or **Path Product** of the segment names.
  - For example, if X and Y are defined as X=abcde,Y=fghij,then the path corresponding to X followed by Y is denoted by

XY=abcdefghij

  - 
  - Similarly,

YX=fghijabcde

aX=aabcde

Xa=abcdea

XaX=abcdeaabcde

  - 
    - If X and Y represent sets of paths or path expressions, their product represents the set of paths that can be obtained by following every element of X by any element of Y in all possible ways. For example,

X = abc + def + ghi

Y = uvw + z

    - Then,

XY = abcuvw + defuvw + ghiuvw + abcz + defz + ghiz

    - 
      - If a link or segment name is repeated, that fact is denoted by an exponent. The exponent's value denotes the number of repetitions:

$a^1 = a$; $a^2 = aa$; $a^3 = aaa$; $a^n = aaaa \ldots$ n times.

      - Similarly, if

X = abcde

      - then

$X^1 = abcde$

$X^2$ = abcdeabcde = $(abcde)^2$

$X^3$ = abcdeabcdeabcde = $(abcde)^2$abcde

= abcde$(abcde)^2$ = $(abcde)^3$

o

- o The path product is not commutative (that is XY!=YX).
  - o The path product is Associative.

RULE 1: A(BC)=(AB)C=ABC

- o where A,B,C are path names, set of path names or path expressions.
- o The zeroth power of a link name, path product, or path expression is also needed for completeness. It is denoted by the numeral "1" and denotes the "path" whose length is zero - that is, the path that doesn't have any links.

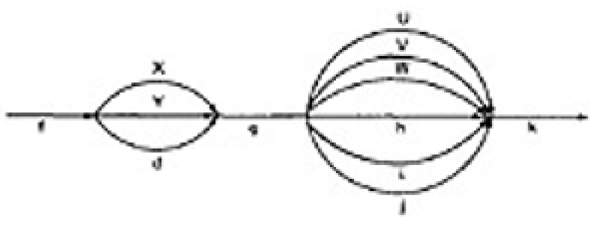
$a^0 = 1$

$X^0 = 1$

o

**PATH SUMS:**

- o The "+" sign was used to denote the fact that path names were part of the same set of paths.
- o The "PATH SUM" denotes paths in parallel between nodes.
- o Links a and b in Figure 5.1a are parallel paths and are denoted by a + b. Similarly, links c and d are parallel paths between the next two nodes and are denoted by c + d.
- o The set of all paths between nodes 1 and 2 can be thought of as a set of parallel paths and denoted by eacf+eadf+ebcf+ebdf.
  - o If X and Y are sets of paths that lie between the same pair of nodes, then X+Y denotes the UNION of those set of paths. For example, in Figure 5.2:



o

*Figure 5.2: Examples of path sums.*

- o The first set of parallel paths is denoted by X + Y + d and the second set by U + V + W + h + i + j. The set of all paths in this flowgraph is f(X + Y + d)g(U + V + W + h + i + j)k

- The path is a set union operation, it is clearly Commutative and Associative.

RULE 2: X+Y=Y+X

RULE 3: (X+Y)+Z=X+(Y+Z)=X+Y+Z

o

## What is Syntax Testing?

Syntax Testing, a black box testing technique, involves testing the System inputs and it is usually automated because syntax testing produces a large number of tests. Internal and external inputs have to conform the below formats:

Format of the input data from users.

File formats.

Database schemas.

## Syntax Testing - Steps:

Identify the target language or format.

Define the syntax of the language.

Validate and Debug the syntax.

## Syntax Testing - Limitations:

Sometimes it is easy to forget the normal cases.

Syntax testing needs driver program to be built that automatically sequences through a set of test cases usually stored as data.

## What is Decision Table Testing?

Decision table testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form. That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage.

A **Decision Table** is a tabular representation of inputs versus rules/cases/test conditions. It is a very effective tool used for both complex software testing and requirements management. A decision table helps to check all possible combinations of conditions for testing and testers can also identify missed conditions easily. The conditions are indicated as True(T) and False(F) values.
Let's learn with an example.

### Example 1: How to make Decision Base Table for Login Screen

Let's create a decision table for a login screen.

The condition is simple if the user provides the correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

| Conditions | Rule 1 | Rule 2 | Rule 3 | Ru |
|---|---|---|---|---|
| **Username (T/F)** | F | T | F | T |
| **Password (T/F)** | F | F | T | T |
| **Output (E/H)** | E | E | E | H |

**Legend:**

**T –** Correct username/password
**F –** Wrong username/password
**E –** Error message is displayed
**H –** Home screen is displayed

**Interpretation:**

**Case 1 –** Username and password both were wrong. The user is shown an error message.
**Case 2 –** Username was correct, but the password was wrong. The user is shown an error message.
**Case 3 –** Username was wrong, but the password was correct. The user is shown an error message.
**Case 4 –** Username and password both were correct, and the user navigated to the homepage

While converting this to a test case, we can create 2 scenarios,

> Enter the correct username and correct password and click on login, and the expected result will be the user should be navigated to the homepage

And one from the below scenario

> Enter wrong username and wrong password and click on login, and the expected result will be the user should get an error message
> Enter correct username and wrong password and click on login, and the expected result will be the user should get an error message
> Enter wrong username and correct password and click on login, and the expected result will be the user should get an error message

As they essentially test the same rule.

## Why Decision Table Testing is Important?

**Decision Table Testing is Important** because it helps to test different combinations of conditions and provides better test coverage for complex business logic. When testing the behavior of a large set of inputs where system behavior differs with each set of inputs, decision table testing provides good coverage and the representation is simple so it is easy to interpret and use.
In Software Engineering, boundary value and equivalent partition are other similar techniques used to ensure better coverage. They are used if the system shows the **same** behavior for a large set of inputs. However, in a system where for each set of input values the system behavior is **different**, boundary value and equivalent partitioning technique are not effective in ensuring good test coverage.

In this case, decision table testing is a good option. This technique can make sure of good coverage, and the representation is simple so that it is easy to interpret and use.

## What is State Transition Testing?

State Transition testing, a black box testing technique, in which outputs are triggered by changes to the input conditions or changes to 'state' of the system. In other words, tests are designed to execute valid and invalid state transitions.

## When to use?

When we have sequence of events that occur and associated conditions that apply to those events

When the proper handling of a particular event depends on the events and conditions that have occurred in the past

It is used for real time systems with various states and transitions involved

## Deriving Test cases:

Understand the various state and transition and mark each valid and invalid state

Defining a sequence of an event that leads to an allowed test ending state

Each one of those visited state and traversed transition should be noted down

Steps 2 and 3 should be repeated until all states have been visited and all transitions traversed

For test cases to have a good coverage, actual input values and the actual output values have to be generated

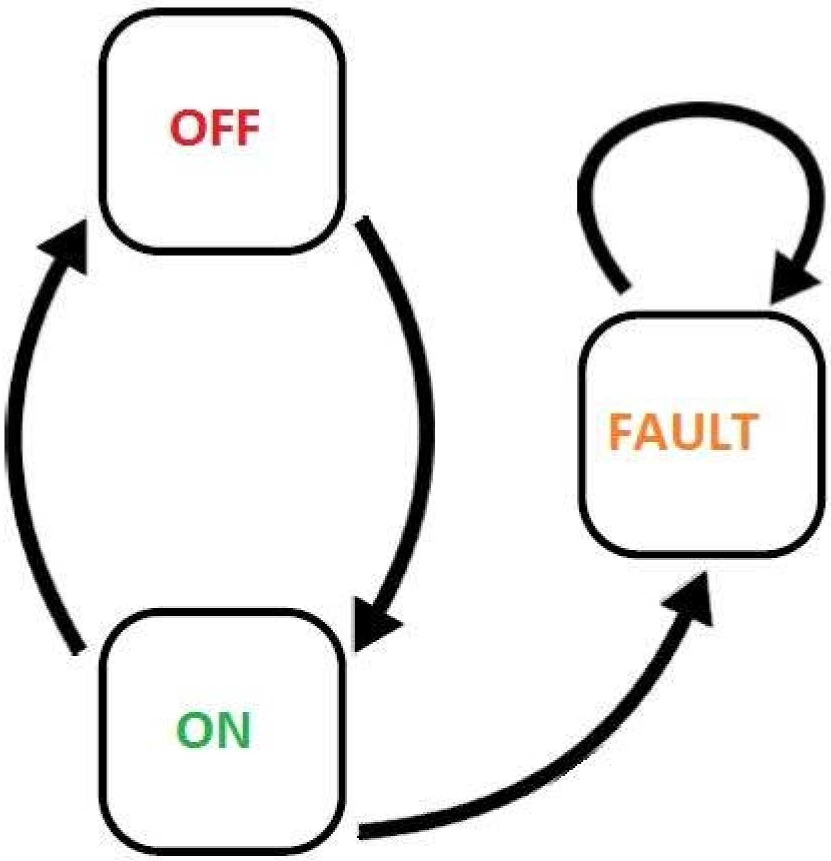
## Advantages:

Allows testers to familiarise with the software design and enables them to design tests effectively.

It also enables testers to cover the unplanned or invalid states.

## Example:

A System's transition is represented as shown in the below diagram:



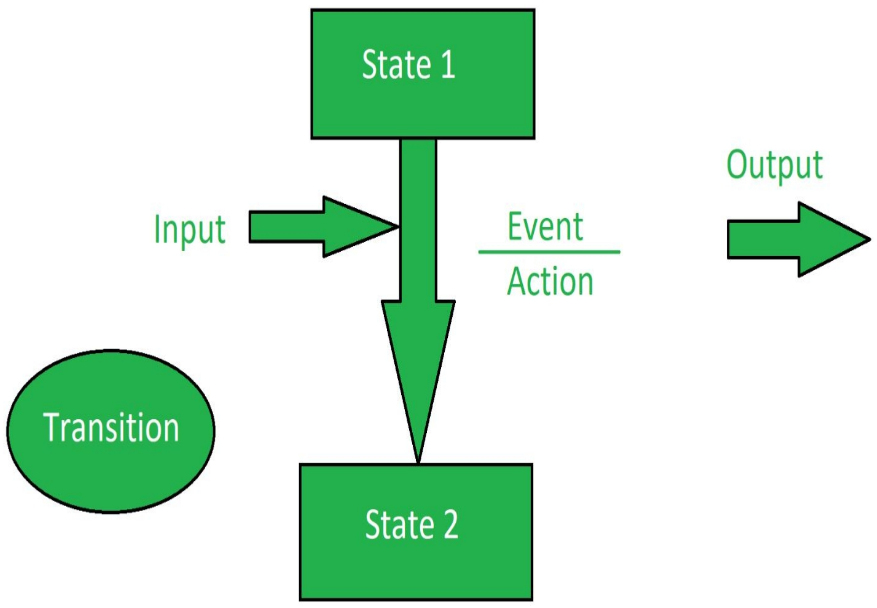
The tests are derived from the above state and transition and below are the possible scenarios that need to be tested.

| Tests | Test 1 | Test 2 | Test 3 |
|---|---|---|---|
| Start State | Off | On | On |
| Input | Switch ON | Switch Off | Switch off |
| Output | Light ON | Light Off | Fault |
| Finish State | ON | OFF | On |

**State Transition Testing** is a type of software testing which is performed to check the change in the state of the application under varying input. The condition of input passed is changed and the change in state is observed.

State Transition Testing is basically a black box testing technique that is carried out to observe the behavior of the system or application for different input conditions passed in a sequence. In this type of testing, both positive and negative input values are provided and the behavior of the system is observed.

State Transition Testing is basically used where different system transitions are needed to be tested.



**Objectives of State Transition Testing:**
The objective of State Transition testing is:

- To test the behavior of the system under varying input.
- To test the dependency on the values in the past.
- To test the change in transition state of the application.
- To test the performance of the system.

**Transition States:**

**Change Mode:**
When this mode is activated then the display mode moves from TIME to DATE.
**Reset:**
When the display mode is TIME or DATE, then reset mode sets them to ALTER TIME or ALTER DATE respectively.
**Time Set:**
When this mode is activated, display mode changes from ALTER TIME to TIME.
**Date Set:**
When this mode is activated, display mode changes from ALTER DATE to DATE.

**State Transition Diagram:**
State Transition Diagram shows how the state of the system changes on

certain inputs.
It has four main components:

1. States
2. Transition
3. Events
4. Actions

**Advantages of State Transition Testing:**

State transition testing helps in understanding the behavior of the system.

State transition testing gives the proper representation of the system behavior.

State transition testing covers all the conditions.

**Disadvantages of State Transition Testing:**

State transition testing can not be performed everywhere.

State transition testing is not always reliable.