RG Kedia College of Commerce

DEPARTMENT OF MCA

COURSE MATERIAL

ON

SOFTWARE QUALITY TESTING

MCA II YEAR III SEMESTER

VISION

- To establish a pedestal for the integral innovation, team spirit, originality and competence in the students, expose them to face the global challenges and become technology leaders of Indian vision of modern society.

MISSION

- To become a model institution in the fields of Engineering, Technology and Management.
- To impart holistic education to the students to render them as industry readyengineers.
- To ensure synchronization of RG kEDIA ideologies with challenging demands of International Pioneering Organizations.

QUALITY POLICY

- To implement best practices in Teaching and Learning process for both UG and PG courses meticulously.
- To provide state of art infrastructure and expertise to impart quality education.
- To groom the students to become intellectually creative and professionallycompetitive.
- To channelize the activities and tune them in heights of commitment and sincerity,the requisites to claim the never - ending ladder of **SUCCESS** year after year.

VISION

To become an innovative knowledge center in MCA through state-of- the-art teaching-learning and research practices, promoting creative thinking professionals.

MISSION

The Department of MCA is dedicated for transforming the students into highly competent Mechanical engineers to meet the needs of the industry, in a changing and challenging technical environment, by strongly focusing in the fundamentals of engineering sciences for achieving excellent results in their professional pursuits.

UNIT 1

INTRODUCTION

Unit -1

INTRODUCTI

ON

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Software project management has wider scope than software engineering process as it involves communication, pre and post delivery support etc.

This tutorial should provide you basic understanding of software product, software design and development process, software project management and design complexities etc. At the end of the tutorial you should be equipped with well understanding of software engineering concepts.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Definitions

IEEE defines software engineering as:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
(2) The study of approaches as in the above statement.

Fritz Bauer, a German computer scientist, defines software engineering as:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

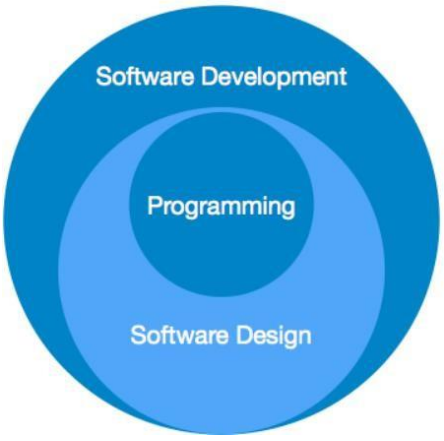
Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software Paradigms



Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes –

- Coding
- Testing
- Integration

Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software -** It is easier to build a wall than to a house or building, likewise, asthe size of software become large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

Characteristics of good software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

This aspect briefs about how well software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on- time software products.

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product. SDLC is a process that consists of a series of planned activities to develop or alter the Software Products.

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality

software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

SDLC is a process followed for a software project, within a software organization. It consistsof a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models. Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

Other related methodologies are Agile Model, RAD Model, Rapid Application Developmentand Prototyping Models.

The Waterfall Model was the first Process Model to be introduced. It is also referred to asa **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

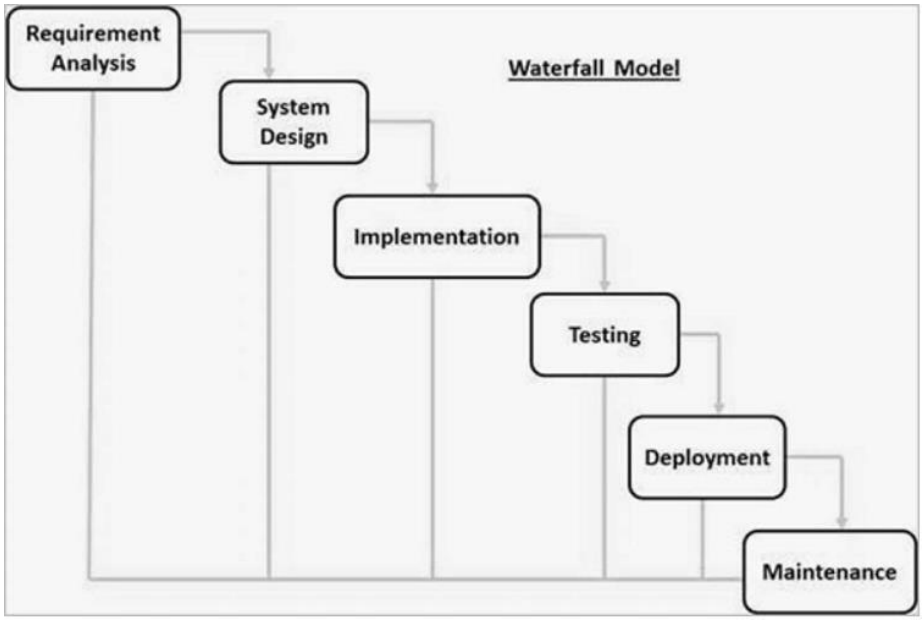
The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entiresystem is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.

- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.

Spiral Model - Design

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design

The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.

Construct or Build

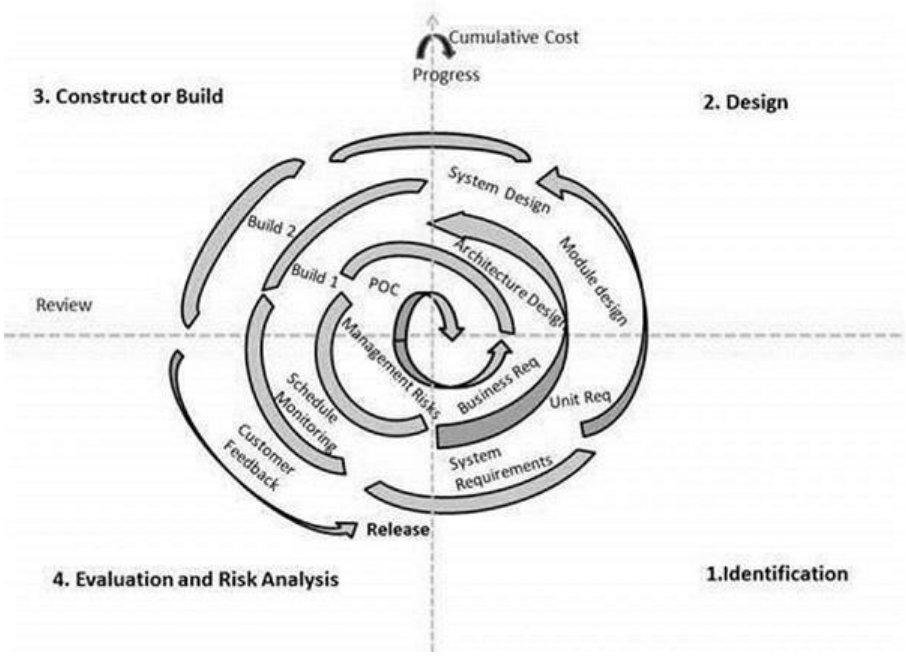
The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

The following illustration is a representation of the Spiral Model, listing the activities in each phase.



Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

Spiral Model Application

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model −

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.

- Customer is not sure of their requirements which are usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

Spiral Model - Pros and Cons

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and thereis a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The advantages of the Spiral SDLC Model are as follows −

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows −

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**.

The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.

Business Requirement Analysis

This is the first phase in the development cycle where the product requirements are understood from the customer's perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The **acceptance test design planning** is done at this stage as business requirements can be used as an input for acceptance testing.

System Design

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

Architectural Design

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**.

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**. It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

The different Validation Phases in a V-Model are explained in detail below.

Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

V- Model – Application

V- Model application is almost the same as the waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly a disciplined domain.

The following pointers are some of the most suitable scenarios to use the V-Model application.

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

V-Model - Pros and Cons

The advantage of the V-Model method is that it is very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today's dynamic world, it becomes very expensive to make the change.

The advantages of the V-Model method are as follows −

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

The disadvantages of the V-Model method are as follows −

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

The Software Prototyping refers to building software application prototypes which displays the functionality of the product under development, but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

What is Software Prototyping?

Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Following is a stepwise approach explained to design a software prototype.

Basic Requirement Identification

This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

Developing the initial Prototype

The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed. While, the workarounds are used to give the same look and feel to the customer in the prototype developed.

Review of the Prototype

The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

Revise and Enhance the Prototype

The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like – time and budget constraints and technical feasibility of the actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until the customer expectations are met.

Prototypes can have horizontal or vertical dimensions. A Horizontal prototype displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions. A Vertical prototype on the other side is a detailed elaboration of a specific function or a sub system in the product.

The purpose of both horizontal and vertical prototype is different. Horizontal prototypes are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market. Vertical prototypes are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system.

Software Prototyping - Types

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely −

Throwaway/Rapid Prototyping

Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

Evolutionary Prototyping

Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is

built. By using evolutionary prototyping, the well-understood requirements are included in the prototype and the requirements are added as and when they are understood.

Incremental Prototyping

Incremental prototyping refers to building multiple functional prototypes of the various sub-systems and then integrating all the available prototypes to form a complete system.

Extreme Prototyping

Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the HTML format. Then the data processing is simulated using a prototype services layer. Finally, the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

Software Prototyping - Application

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed.

Software that involves too much of data processing and most of the functionality is internal with very little user interface does not usually benefit from prototyping. Prototype development could be an extra overhead in such projects and may need lot of extra efforts.

Software Prototyping - Pros and Cons

Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final software developed. The model has its own pros and cons discussed as follows.

The advantages of the Prototyping Model are as follows −

- Increased user involvement in the product even before its implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

The Disadvantages of the Prototyping Model are as follows −

- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

This tutorial will give you a basic understanding on software testing, its types, methods, levels, and other related terminologies.

Why to Learn Software Testing?

In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities −

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

Applications of Software Testing

- **Cost Effective Development** - Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.
- **Product Improvement** - During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.
- **Test Automation** - Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.
- **Quality Check** - Software testing helps in determining following set of properties ofany software such as
    - Functionality
    - Reliability
    - Usability
    - Efficiency
    - Maintainability
    - Portability

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities −

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

When to Start Testing?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC −

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that software is 100% tested. The following aspects are to be considered for stopping the testing process −

- Testing Deadlines
- Completion of test case execution
- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management decision

Roles and Responsibilities of a Software Tester

Test lead/manager: A test lead is responsible for:

- Defining the testing activities for subordinates – testers or test engineers.
- All responsibilities of test planning.
- To check if the team has all the necessary resources to execute the testing activities.
- To check if testing is going hand in hand with the software development in all phases.
- Prepare the status report of testing activities.
- Required Interactions with customers.
- Updating project manager regularly about the progress of testing activities.

Test engineers/QA testers/QC testers are responsible for:

- To read all the documents and understand what needs to be tested.
- Based on the information procured in the above step decide how it is to be tested.
- Inform the test lead about what all resources will be required for software testing.
- Develop test cases and prioritize testing activities.
- Execute all the test case and report defects, define severity and priority for each defect.
- Carry out regression testing every time when changes are made to the code to fix defects.

Overview of Software Engineering Team

How a software application shapes up during the development process entirely depends on the how the software engineering team organizes work and implements various methodologies. For an application to develop properly, it is important that all processes incorporated during the software development are stable and sustainable. Many times developers come under pressure as the delivery date approaches closer this often affects the quality of the software. Rushing through the processes to finish the project on time will only produce a software application which has no or minimal use for the customers. Hence, work organization and planning is important and sticking to the plan is very important. The project manager should ensure that there are no obstacles in the development process and if at all there is an issue it must be resolved with immediate attention.

Overview of Software Testing Team

How soon and how well you can achieve your testing goals depends solely on the capabilities of the testing team. Within the testing team itself it is important to have the correct blend of testers who can efficiently work together to achieve the common testing goals. While forming a team for testing, it is important to ensure that the members of the team jointly have a combination of all the relevant domain knowledge that is required to test the software under development.

It is very important to ensure that the software testing team has a proper structure. The hierarchy and roles should be clearly defined and responsibilities too should be well defined and properly distributed amongst the team members. When the team is well organized the

work can be handled well. If every team member knows what duties he or she has to perform then they will be able to finish their duties as required well within the time limit. It is important to keep track of the testers' performance. It is very important to check what kind of defects the tester is able to uncover and what kind of detects he tends to miss. This will give you a fair idea about how serious your team is about the work.

All the team members should work together to prepare a document that clearly defines the roles and responsibilities of all the team members. Once the document is prepared the role of each member should be communicated clearly to everyone. Once the team members are clear about who is going to handle which area of the project, then in case of any issue it will be easy to determine who needs to be contacted.

Each member of the team should be provided with the necessary documents that provide information on how the task would be organized, what approach will be followed, how things are scheduled, how many hours have been allocated to each member and all details related to applicable standards and quality processes.

Software Tester Role

A Software tester (software test engineer) should be capable of designing test suites and should have the ability to understand usability issues. Such a tester is expected to have sound knowledge of software test design and test execution methodologies. It is very important for a software tester to have great communication skills so that he can interact with the development team efficiently. The roles and responsibilities for a usability software tester are as follows:

1. A Software Tester is responsible for designing testing scenarios for usability testing.
2. He is responsible for conducting the testing, thereafter analyze the results and then submit his observations to the development team.
3. He may have to interact with the clients to better understand the product requirements or in case the design requires any kind of modifications.
4. Software Testers are often responsible for creating test-product documentation and also has to participate in testing related walk through.

A software tester has different sets of roles and responsibilities. He should have in depth knowledge about software testing. He should have a good understanding about the system which means technical (GUI or non-GUI human interactions) as well as functional product aspects. In order to create test cases it is important that the software tester is aware of various testing techniques and which approach is best for a particular system. He should know what are various phases of software testing and how testing should be carried out in each phase. The responsibilities of the software tester include:

1. Creation of test designs, test processes, test cases and test data.
2. Carry out testing as per the defined procedures.
3. Participate in walkthroughs of testing procedures.
4. Prepare all reports related to software testing carried out.
5. Ensure that all tested related work is carried out as per the defined standards and procedures.

Software Test Manager Role

Managing or leading a test team is not an easy job. The company expects the test manager to know testing methodologies in detail. A test manager has to take very important decisions regarding the testing environment that is required, how information flow would be managed and how testing procedure would go hand in hand with development. He should have sound knowledge about both manual as well as automated testing so that he can decide how both the methodologies can be put together to test the software. A test manager should have sound knowledge about the business area and the client's requirement, based on that he should be able to design a test strategy, test goal and objectives. He should be good at project planning, task and people coordination, and he should be familiar with various types of testing tools. Many people get confused between the roles and responsibilities of a test manager and test lead.For a clarification, a test lead is supposed to have a rich technical experience which includes, programming, handling database technologies and various operating systems, whereas he may not be as strong as Software Test Manager regarding test project management and coordination. The responsibilities of the test manager are as follows:

1. Since the test manager represents the team he is responsible for all interdepartmental meetings.
2. Interaction with the customers whenever required.
3. A test manager is responsible for recruiting software testing staff. He has to supervise all testing activities carried out by the team and identify team members who require more training.
4. Schedule testing activities, create budget for testing and prepare test effort estimations.
5. Selection of right test tools after interacting with the vendors. Integration of testing and development activities.
6. Carry out continuous test process improvement with the help of metrics.
7. Check the quality of requirements, how well they are defined.
8. Trace test procedures with the help of test traceability matrix.

Software Test Automator Role

Software test automator or an automated test engineer should have very good understanding of what he needs to test- GUI designs, load or stress testing. He should be proficient in automation of software testing, and he should be able to design test suites accordingly. A software test automator should be comfortable using various kinds of automation tools and should be capable of upgrading their skills with changing trends. He should also have programming skills so that he is able to write test scripts without any issues. The responsibilities of a tester at this position are as follows:

1. He should be able to understand the requirement and design test procedures and test cases for automated software testing.
2. Design automated test scripts that are reusable.
3. Ensure that all automated testing related activities are carried out as per the standards defined by the company.

Interactions between Software Test Team And Business Teams

If at all a customer has any issues related to testing activities and operational matters of the project then it is the software testing manager who is responsible for communicating the details to the client regarding how things are being managed. The software testing manager

not only answers the queries of the customers but also ensures that the project is completed on time as per the requirement of the customer.

Interactions between Software Test Team And Development Teams

In order to produce good software applications, it is important that software testing and software development teams work together with good understanding. For this it is important that the testers and developers are comfortable with each other's role and understand well that they have a common goal and it is wise to listen each other. A good communication skill is very important both for testers and developers.

Before getting started with testing work it is important to discuss the basic guidelines and expectations so that there is no confusion in later stages. Criticism should be taken in a positive sense. It is important to understand that developers and testers have a common goal of producing high quality software. A tester is not discovering bugs to show someone down, the idea is to learn from mistakes and avoid repeating them in future. A culture of constructive criticism can be of great help.

Interactions between Software Test Team And Release Management Teams

The release management teams are responsible for moving the software from development into production. This team is responsible for planning the releases for hardware, software and testing. It is also responsible for development of software development procedures and for coordinating interactions and training of releases. Software testing is considered to be a very important aspect of software engineering life cycle but it does not get over with development. Testing and verification is a very important part of release management exercise.

Interactions between Software Test Manager And Software Project Manager

The job of a software test manager is not an easy one. He has to recruit testing team and take responsibility for getting them trained. A software manager has to perform ongoing analysis of various testing processes and ensure that the testing team is carrying out all the processes correctly. This job is of great responsibility as the software testing manager is the one who selects, introduces and implement various tools for testing. A software test manager is responsible for finalizing templates for testing documents, test reports and other procedures.

Since a software tester manager has to deal with all the details of various testing activities, it is very important for him to be in constant touch with the project manager and provide necessary support in project planning and scheduling so that the project can be successfully completed in time within the specified financial budget limits.

Verification & Validation

These two terms are very confusing for most people, who use them interchangeably. The following table highlights the differences between verification and validation.

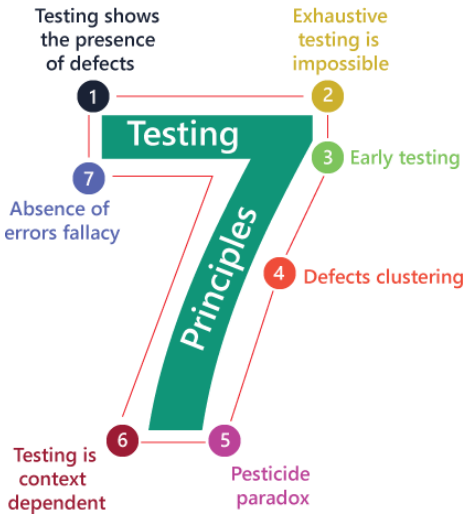
Sr.No.	Verification	Validation
1	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
2	Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behavior.

3	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overallproduct.
4	Done by developers.	Done by testers.
5	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verifya software.	It has dynamic activities, as it includes executing the software against the requirements.
6	It is an objective process and no subjective decision should be needed to verify a software.	It is a subjective process and involves subjective decisions on how well a software works.

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. Here, in this section, we are going to learn about the seven essential principles of software testing.

Let us see the seven different testing principles, one by one:

- Testing shows the presence of defects
- Exhaustive Testing is not possible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context-dependent
- Absence of errors fallacy



The test engineer will test the application to make sure that the application is bug or defects free. While doing testing, we can only identify that the application or software has any errors. The primary purpose of doing testing is to identify the numbers of unknown bugs with the help of various methods and testing techniques because the entire test should be traceable to the customer requirement, which means that to find any defects that might cause the product failure to meet the client's needs.

By doing testing on any application, we can decrease the number of bugs, which does not mean that the application is defect-free because sometimes the software seems to be bug- free while performing multiple types of testing on it. But at the time of deployment in the production server, if the end-user encounters those bugs which are not found in the testing process.

Exhaustive Testing is not possible

Sometimes it seems to be very hard to test all the modules and their features with effective and non- effective combinations of the inputs data throughout the actual testing process.

Hence, instead of performing the exhaustive testing as it takes boundless determinations and most of the hard work is unsuccessful. So we can complete this type of variations according to the importance of the modules because the product timelines will not permitus to perform such type of testing scenarios.

Early Testing

Here early testing means that all the testing activities should start in the early stages of the software development life cycle's **requirement analysis stage** to identify the defects because if we find the bugs at an early stage, it will be fixed in the initial stage itself, which may cost us very less as compared to those which are identified in the future phase of the testing process.

To perform testing, we will require the requirement specification documents; therefore, if the requirements are defined incorrectly, then it can be fixed directly rather than fixing them in another stage, which could be the development phase.

Defect clustering

The defect clustering defined that throughout the testing process, we can detect the numbers of bugs which are correlated to a small number of modules. We have variousreasons for this, such as the modules could be complicated; the coding part may be complex, and so on.

These types of software or the application will follow the **Pareto Principle**, which states that we can identify that approx. Eighty percent of the complication is present in 20 percent of the modules. With the help of this, we can find the uncertain modules, but this method has its difficulties if the same tests are performing regularly, hence the same test will not able to identify the new defects.

Pesticide paradox

This principle defined that if we are executing the same set of test cases again and again over a particular time, then these kinds of the test will not be able to find the new bugs in the software or the application. To get over these pesticide paradoxes, it is very significant to review all the test cases frequently. And the new and different tests are necessary to be written for the implementation of multiple parts of the application or the software, which helps us to find more bugs.

Testing is context-dependent

Testing is a context-dependent principle states that we have multiple fields such as e-commerce websites, commercial websites, and so on are available in the market. There is a definite way to test the commercial site as well as the e-commerce websites because every application has its own needs, features, and functionality. To check this type of application, we will take the help of various kinds of testing, different technique, approaches, and multiple methods. Therefore, the testing depends on the context of the application.

Absence of errors fallacy

Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free. But there is the chance when the application is tested beside the incorrect requirements, identified the flaws, and fixed them on a given period would not help as testing is done on the wrong specification, which does not apply to the client's requirements. The absence of error fallacy means identifying and fixing the bugs would not help if the application is impractical and not able to accomplish the client's requirements and needs.

Unit 2

Software Testing
Requirement

Software Requirement

It's a primary requirement needed in the development of a software product. These requirements works as a base and is being used in developing a particular software product to perform specifically for a targeted group or audience and for the specific environment.

Basically, these requirements imparts the appearance or an overview of a desired software product such as how will it look its functionality, features, how it should perform, etc. based on which, developer carries out the development of a software development.

These requirements are of very much importance as any sort of compromise to them may produce undesirable final product and may fail to meet the needs & expectations of a clientor a user. Therefore, there exists a separate phase in a **SDLC** to gather, study and analyse the software requirements so as to avoid such type of circumstances.

Types of Requirements in Software Testing

Further, these requirements may also be categorized into multiple types, based on different perspective. Let’s go through each of them.

Business Requirements:

These requirements are specified from the business point of view. It generally involves the specified objectives and goals of a particular project that needs to be fulfilled. It provides an abstract of a project. These requirements are not meant for specifying the functionalities or technicalities of a desired software product rather it outlines a general overview of a product, such as its primary use, why it is needed, its scope & vision, what business benefits will be gain, intended audience or users, etc. It generally involves the participation of the client, stakeholders, business and project managers for gathering and analyzing the business requirements.

Through business requirements, it is easy to assess the project cost, time required, business risks involved and many such things associated with a software development project.

System Requirements:

Requirements to be incorporated in a software product under development to make a software product perform and function in a specific manner to achieve a specific target and

goal falls under the category of system requirements. These system requirements may be broadly classified in two types **functional requirements and non-functional requirements**.

Functional requirements:

Requirements encompassing the functional attributes and behaviour of a software product are called functional requirements. These requirements reflect the working and functionalities of an intended software product.

These requirements defines and describes the functions to be performed, and features tobe possessed by a software product. What and how does a product supposed to perform on accepting inputs from the user, and what desirable output it should provide to the users. These requirements should be complete and clearly well-defined so as to meet all the specified feature and functionalities without misunderstanding or leaving the requirement so as to achieve a desirable quality product.

Non-Functional Requirements:

Requirements other than functional requirements which are essential and contribute towards the performance of a software product under variant type of conditions and multiple environments are commonly known as Non-functional requirements. These requirements are used to evaluate and assess the software product behaviour other than its specific or desired behaviour under unexpected conditions and environment, contrary to what is favourable for its functioning. It also covers the standards, rules and regulation thata software product must adhere and conform to it.

These requirements are accountable for system's performance and quality and, generally cover the following requirements and attributes:

Performance.

Response Time.

Throughput.

Utilization.

Efficiency.

Scalability.

Capacity.

Availability

.

Delivery.

Reliability.

Recoverability.

Maintainability.

Serviceability.

Security.

Regulatory.

Manageability.

Environmental.

Data Integrity.

Usability.

Interoperability.

As there is no a specific criterion or rule to assess these non-functional aspects, it becomes a tedious job to verify these attributes of a software product. Hence, it requires the usage of metrics to validate a software product, quantitatively against these requirements.

Further, these requirements are documented in the form of Software Requirement Specification(SRS) by the project manager, which works as a manual for the development team to proceed the development process in a particular and correct direction.

User Requirements:

Requirements generated from a users point of view and scenarios of using a software product in a multiple manner under real environment by a targeted user to execute a particular task, specifies the user requirements. It defines the user's expectation from a software product. As user’s exhaustive needs may not be covered under the domain of system requirement, it may be covered separately by business analysts through studying and analyzing the user requirements.

These types of requirements are generally gathered and documented using use cases, user scenarios, and user stories. These requirements are documented in a user requirement

document (URD) format by making use of narrative text and are usually signed off by the intended users.

Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non- behavioral requirements.
They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Following are the differences between Functional and Non Functional Requirements

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfil the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.

Functional Requirements	Non Functional Requirements
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality ofthe software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing,etc are done.	Non-Functional Testing like Performance, Stress, Usability, Securitytesting, etc are done.
Usually easy to define.	Usually more difficult to define.
Example **1)** Authentication of user whenever he/she logs into the system. **2)** System shutdown in case of a cyber-attack. **3)** A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example** **1)** Emails should be sent with a latency of no greater than 12 hours from such an activity. **2)** The processing of each request should be done within 10 seconds **3)** The site should load in 3 seconds when the number of simultaneous users are > 10000

Advantages of Functional Requirement

- Helps you to check whether the application is providing all the functionalities that were mentioned in the functional requirement of that application
- A functional requirement document helps you to define the functionality of a system or one of its subsystems.
- Functional requirements along with requirement analysis help identify missing requirements. They help clearly define the expected system service and behavior.
- Errors caught in the Functional requirement gathering stage are the cheapest to fix.
- Support user goals, tasks, or activities for easy project management
- Functional requirement can be expressed in Use Case form or user story as they exhibit externally visible functional behavior.

Advantages of Non-Functional Requirement

- The non-functional requirements ensure the software system follow legal and compliance rules.
- They ensure the reliability, availability, and performance of the software system
- They ensure good user experience and ease of operating the software.

- They help in formulating security policy of the software system.

Disadvantages of Non-functional requirement

- None functional requirement may affect the various high-level software subsystem
- They require special consideration during the software architecture/high-level design phase which increases costs.
- Their implementation does not usually map to the specific software sub-system,
- It is tough to modify non-functional once you pass the architecture phase.

Software Testing Review process:

A **software review** is a process or meeting during which a software product is examined by a project personnel, users, computers, user representatives, or other interested parties for comment or approval."

It is very clear from the above definition that a software review is as essential as **software testing**. A software review has its own significance as it provides a better view on the developed software. It requires a team that can provide insights of the build software.

In the review, various things come to surface such as future casualties, technical content, quality, specifications of the software, etc. This deep overview of a software gives an idea that how will the software run and is shortcomings that are likely to occur in the near future. It also provides an idea that this software is even worth launching. Sometimes, the software has fewer benefits and greater number of disadvantages falling, in this condition the software is discarded or a better development of better software is proposed.

The value of the software is defined by the reviews. The qualified team looks over the software as per the guidelines provided to them. The specifications and standards are keptin mind while reviewing build software. It is a healthy form of discussion between the people who are in direct developers of the authors as well as the staff associated.

Types of Review in Software Testing

There are mainly Three Types of Reviews in a Software Testing.

1. **Software Peer Reviews:**
   This type of review is conducted by the main author of the software, or it can be between the colleagues so that the evaluation can be done of the technical content or quality of the work. A discussion is always a solution for a software analysis. Aforementioned, it always provides a deeper insight of the software shortcomings and also its benefits. If the software shortcoming weigh higher than benefits, the software goes in testing mode again and it rectified as required.
   Different types of Software Peer Reviews are there which are enumerated below:
   - **Code Review**: The source code of the software is examined here. The software is checked for the bugs and the bugs are removed from the code.
   - **Pair programming**: It is also a type of code review involving two people. The two people develop a code together at the same workstation.
   - **Inspection**: It is a formal type of review where a person has to go through a defined set of instructions in order to find defect/defects. There can be a number of reviewers involved in this type of reviewing.

- **Walkthrough**: This is a process where the authors of the software as well as other associates are gathered at one place and they discuss about the software defects. Questions are made, comments are given, answers are given to all the queries people have regarding the software. With all the members' satisfaction, conclusions are made.
- **Technical review:** This is the team of qualified personnel examines the suitability of the software product for its intended discrepancies from the specification and standards point of view.

2. **Software Management Reviews:**
   The management representatives are responsible for this type of review. The statusof the work is evaluated and the decision by the activities of the software. This review is very important in making a decision regarding software.

3. **Software Audit Reviews:**
   These are conducted by the personnel outside of the software project. They evaluate the software with specifications, standards, and other criteria.
   All the reviews are important to make software run successfully. Formal code reviews require a huge amount of investment of the time and the preparation of the review. Internal disputes also may carry leverage on these reviews. It is a time taking process and it may not be that accurate.

There are certain guidelines that a person has to follow to perform a perfect review on build software.

- **Defect Prevention** is the main goal of the software as well as look out for the functionality of the software.
- Review of the requirement specifications should be done carefully so as to evaluate the software as per required.
- The list should be clarified.
- Correctness, portability, security and maintainability should be checked in build software.

UNIT 3

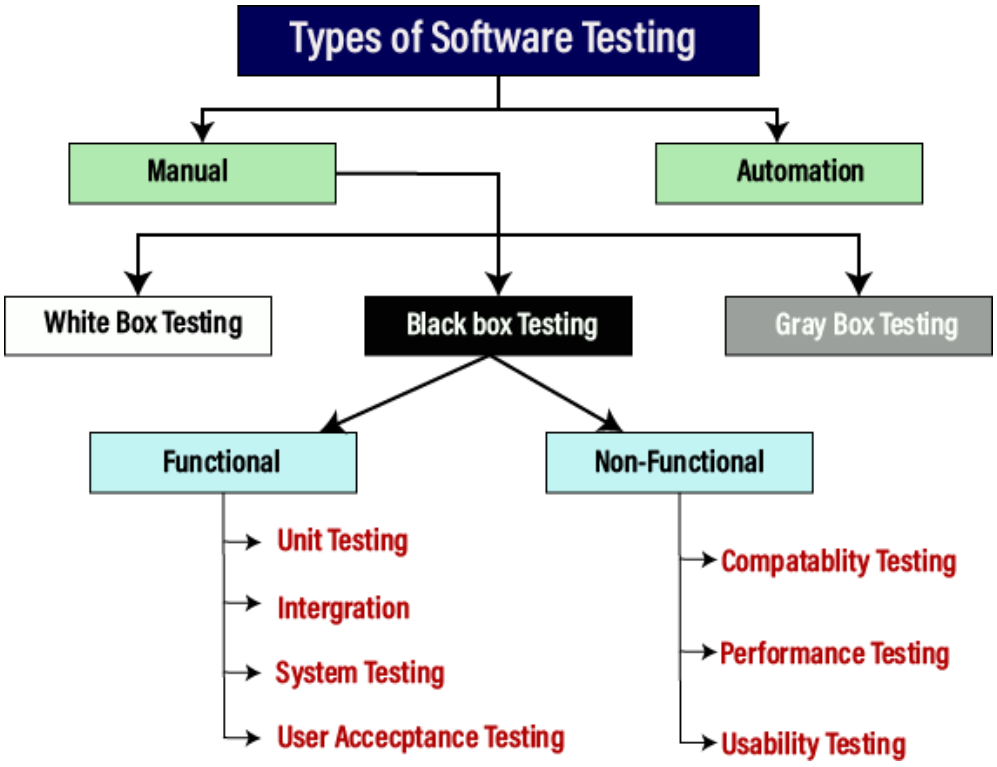
TESTING

TECHNIQUES

Type of Software testing

We have various types of testing available in the market, which are used to test the application or the software.

With the help of below image, we can easily understand the type of software testing:



Manual testing

The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing. While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

Manual testing can be further divided into three types of testing, which are as follows:

- **White box testing**
- **Black box testing**
- **Gray box testing**

Automation testing

Automation testing is a process of converting any manual test cases into the test scripts with the help of automation tools, or any programming language is known as automation testing. With the help of automation testing, we can enhance the speed of our test execution because here, we do not require any human efforts. We need to write a test script and execute those scripts.

White Box Testing

The box testing approach of software testing consists of black box testing and white box testing. We are discussing here white box testing which also known as glass box is **testing, structural testing, clear box testing, open box testing and transparent box testing**. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and identify the bugs and sends it to the developer.

The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.

Here, the test engineers will not include in fixing the defects for the following reasons:

- Fixing the bug might interrupt the other features. Therefore, the test engineershould always find the bugs, and developers should still be doing the bug fixes.
- If the test engineers spend most of the time fixing the defects, then they may be unable to find the other bugs in the application.

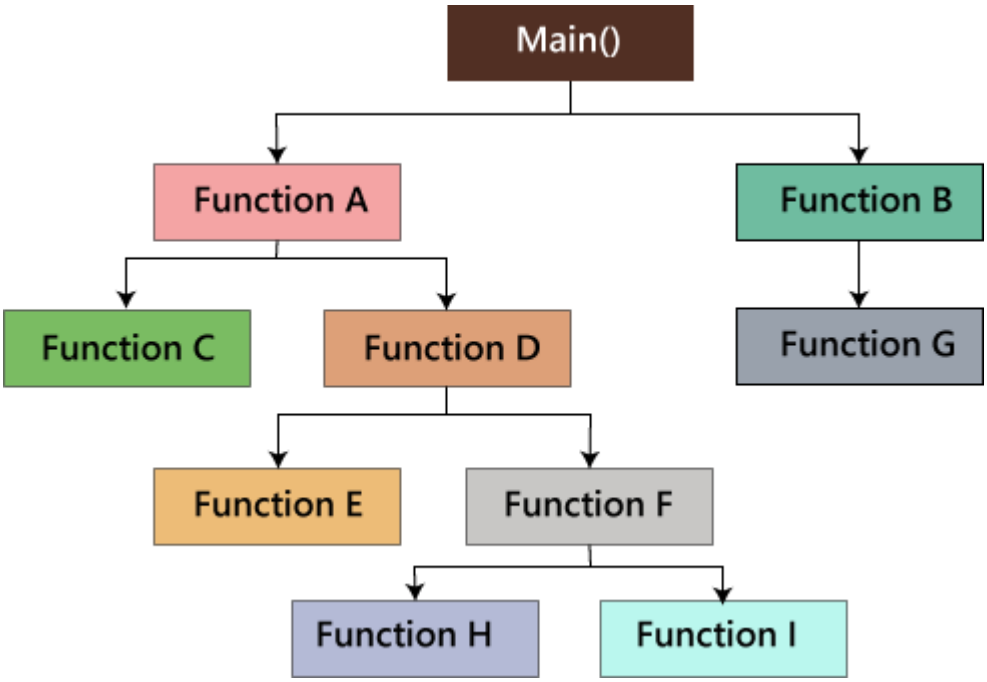
The white box testing contains various tests, which are as follows:

- Path testing
- Loop testing
- Condition testing

- Testing based on the memory perspective
- Test performance of the program

Path testing

In the path testing, we will write the flow graphs and test all independent paths. Here writing the flow graph implies that flow graphs are representing the flow of the programand also show how every program is added with one another as we can see in the belowimage:



And test all the independent paths implies that suppose a path from main() to function G, first set the parameters and test if the program is correct in that particular path, and in thesame way test all other paths and fix the bugs.

Loop testing

In the loop testing, we will test the loops such as while, for, and do-while, etc. and also check for ending condition if working correctly and if the size of the conditions is enough.

For example: we have one program where the developers have given about 50,000 loops.

For example: we have one program where the developers have given about 50,000 loops.

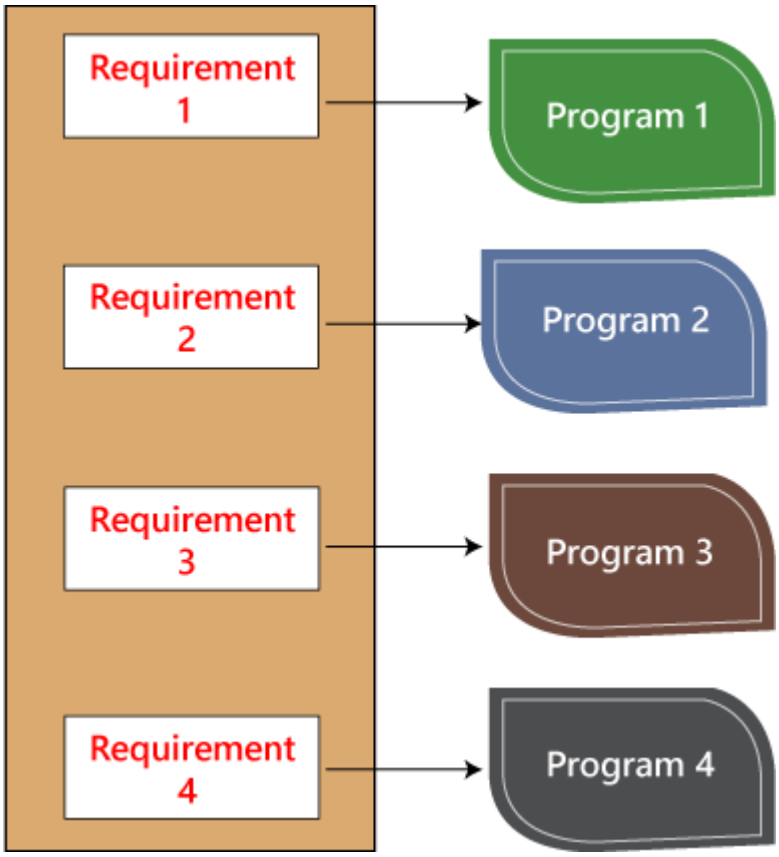
1. {
2. while(50,000)
3. ……

4. ……
5. }

We cannot test this program manually for all the 50,000 loops cycle. So we write a small program that helps for all 50,000 cycles, as we can see in the below program, that test P is written in the similar language as the source code program, and this is known as a Unit test. And it is written by the developers only.

1. Test P
2. {
3. ……
4. …… }

As we can see in the below image that, we have various requirements such as 1, 2, 3, 4. And then, the developer writes the programs such as program 1,2,3,4 for the parallel conditions. Here the application contains the 100s line of codes.

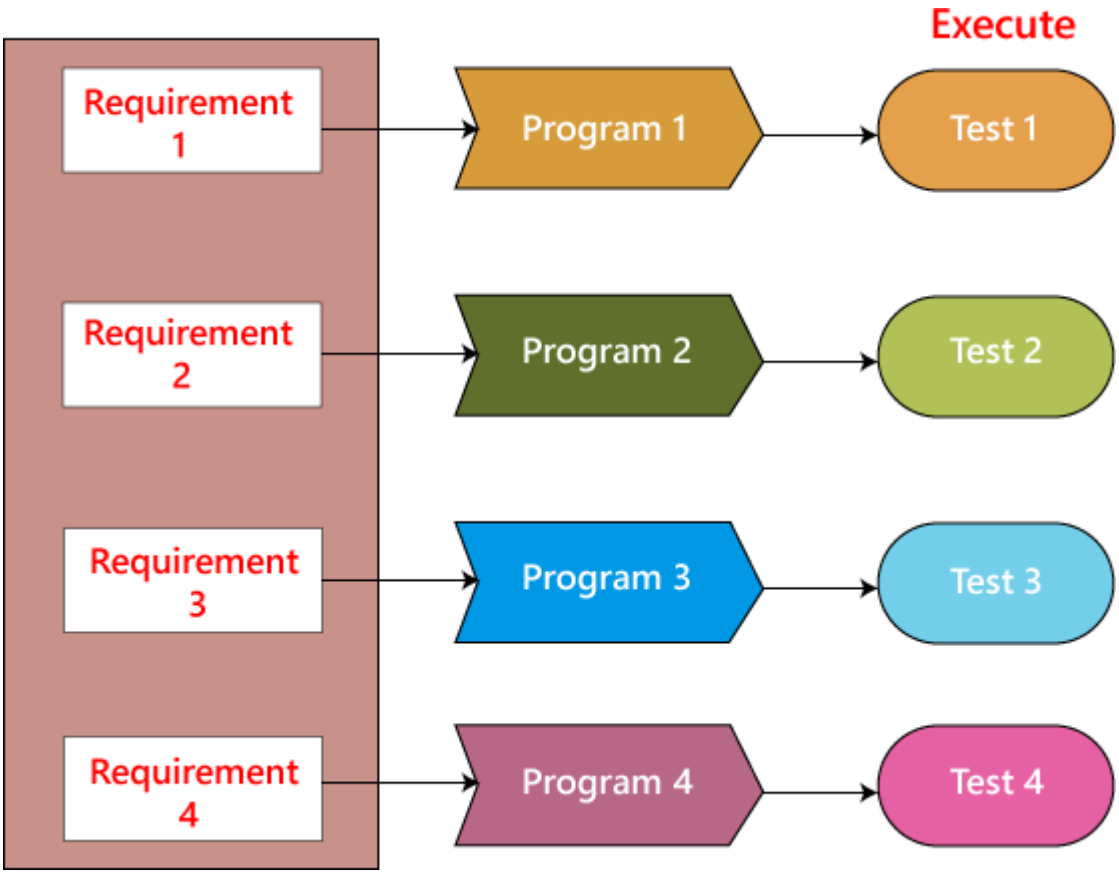


The developer will do the white box testing, and they will test all the five programs line by line of code to find the bug. If they found any bug in any of the programs, they will correct it. And they again have to test the system then this process contains lots of time and effort and slows down the product release time.

Now, suppose we have another case, where the clients want to modify the requirements, then the developer will do the required changes and test all four program again, which takelots of time and efforts.

These issues can be resolved in the following ways:

In this, we will write test for a similar program where the developer writes these test code in the related language as the source code. Then they execute these test code, which is also known as **unit test programs**. These test programs linked to the main program and implemented as programs.



Therefore, if there is any requirement of modification or bug in the code, then the developer makes the adjustment both in the main program and the test program and thenexecutes the test program.

Condition testing

In this, we will test all logical conditions for both **true** and **false** values; that is, we will verify for both **if** and **else** condition.

For example:

1. if(condition) - true

2. {
3. …..
4. ……
5. ……
6. }
7. else - false
8. {
9. …..
10. ……
11. ……
12. }

The above program will work fine for both the conditions, which means that if the conditionis accurate, and then else should be false and conversely.

Testing based on the memory (size) perspective

The size of the code is increasing for the following reasons:

- **The reuse of code is not there**: let us take one example, where we have four programs of the same application, and the first ten lines of the program are similar. We can write these ten lines as a discrete function, and it should be accessible by the above four programs as well. And also, if any bug is there, we can modify the line of code in the function rather than the entire code.
- The **developers use the logic** that might be modified. If one programmer writes code and the file size is up to 250kb, then another programmer could write a similar code using the different logic, and the file size is up to 100kb.
- The **developer declares so many functions and variables** that might never be usedin any portion of the code. Therefore, the size of the program will increase.

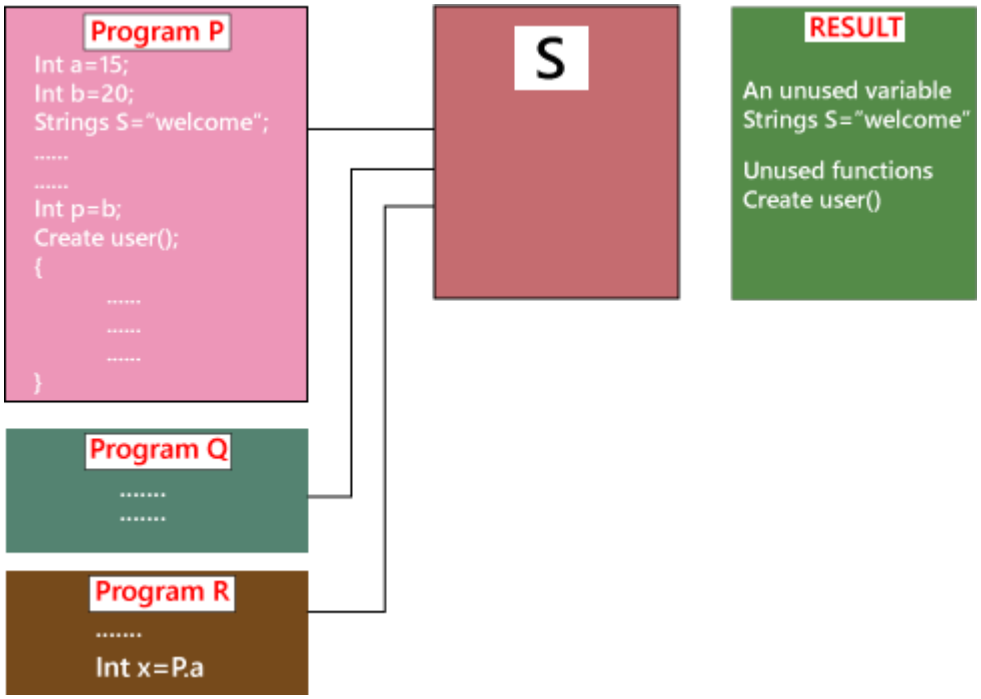
For example,

```
1. Int a=15;
2. Int b=20;
3. String S=
"Welcome";4. ….
5. …..
6. …..
7. ….
```

8. …..
9. Int p=b;
10. Create user()
11. {
12. ……
13. ……
14. 200's line of code
15. }

In the above code, we can see that the **integer a** has never been called anywhere in the program, and also the function **Create user** has never been called anywhere in the code. Therefore, it leads us to memory consumption.

We cannot remember this type of mistake manually by verifying the code because of the large code. So, we have a built-in tool, which helps us to test the needless variables and functions. And, here we have the tool called **Rational purify**.



Suppose we have three programs such as Program P, Q, and R, which provides the input to S. And S goes into the programs and verifies the unused variables and then gives the outcome. After that, the developers will click on several results and call or remove the unnecessary function and the variables.

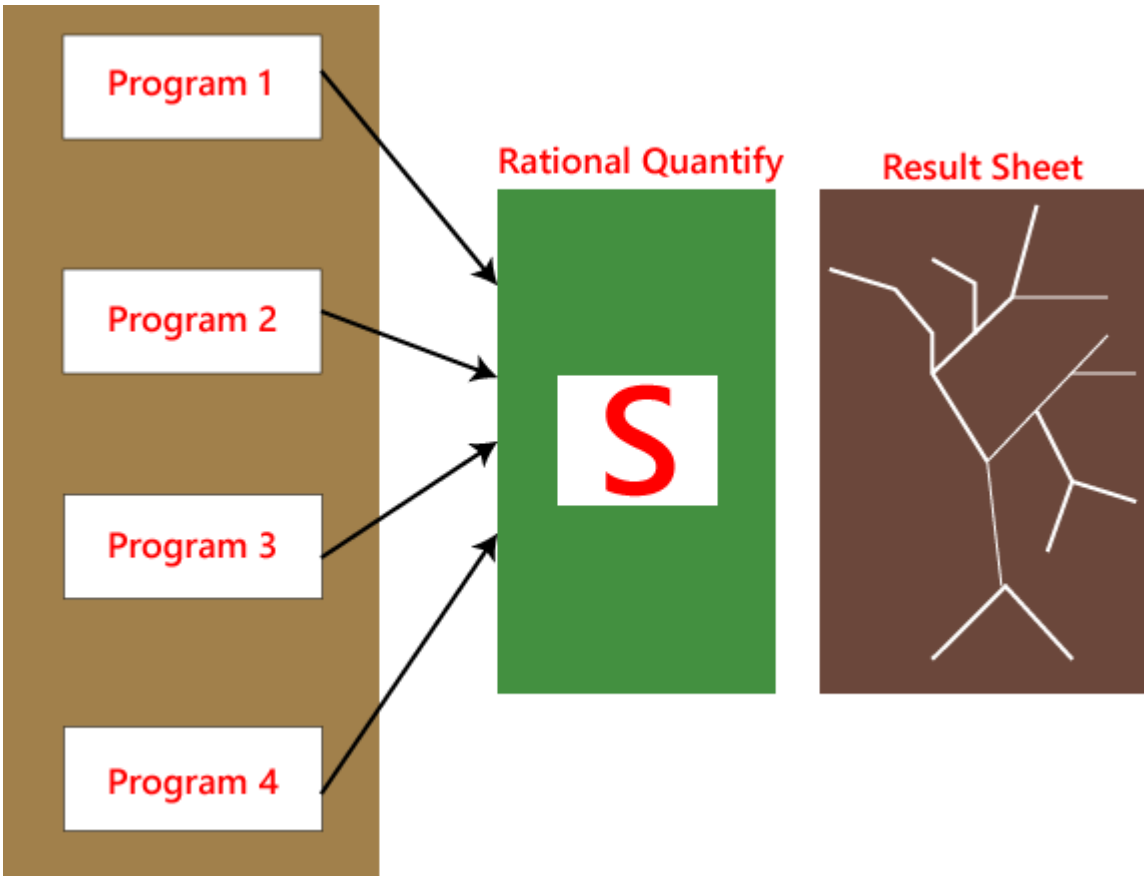
This tool is only used for the C programming language and C++ programming language; foranother language, we have other related tools available in the market.

- The developer does not use the available in-built functions; instead they write the full features using their logic. Therefore, it leads us to waste of time and also postpone the product releases.

Test the performance (Speed, response time) of the program

The application could be slow for the following reasons:

- When logic is used.
- For the conditional cases, we will use **or** & **and** adequately.
- Switch case, which means we cannot use **nested if**, instead of using a switch case.



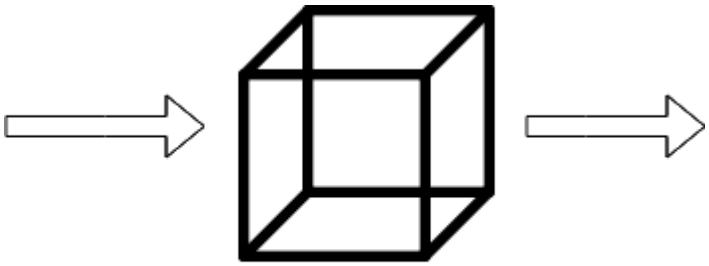
As we know that the developer is performing white box testing, they understand that the code is running slow, or the performance of the program is also getting deliberate. And the developer cannot go manually over the program and verify which line of the code is slowing the program.

To recover with this condition, we have a tool called **Rational Quantify**, which resolves these kinds of issues automatically. Once the entire code is ready, the rational quantify tool

will go through the code and execute it. And we can see the outcome in the result sheet in the form of thick and thin lines.

Here, the thick line specifies which section of code is time-consuming. When we double-click on the thick line, the tool will take us to that line or piece of code automatically, which is also displayed in a different color. We can change that code and again and use this tool.
When the order of lines is all thin, we know that the presentation of the program has enhanced. And the developers will perform the white box testing automatically because it saves time rather than performing manually.

Test cases for white box testing are derived from the design phase of the software development lifecycle. Data flow testing, control flow testing, path testing, branch testing, statement and decision coverage all these techniques used by white box testing as a guideline to create an error-free software.



Whitebox Testing

White box testing follows some working steps to make testing manageable and easy to understand what the next task to do. There are some basic steps to perform white box testing.

Generic steps of white box testing

- Design all test scenarios, test cases and prioritize them according to high priority number.
- This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by various methods and operations andso on.
- In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, interfaces are able to handle all types of data appropriately or not.
- This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.

- In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

Reasons for white box testing

- It identifies internal security holes.
- To check the way of input inside the code.
- Check the functionality of conditional loops.
- To test function, object, and statement at an individual level.

Advantages of White box testing

- White box testing optimizes code so hidden errors can be identified.
- Test cases of white box testing can be easily automated.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.

Disadvantages of White box testing

- White box testing is too much time consuming when it comes to large-scale programming applications.
- White box testing is much expensive and complex.
- It can lead to production error because it is not detailed by the developers.
- White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

Techniques Used in White Box Testing

Data Flow Testing	Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.
Control Flow Testing	Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the

	tester to set the testing path. Test cases represented by the controlgraph of the program.
Branch Testing	Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
Statement Testing	Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source codeat least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.
Decisio n Testing	This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomesfrom the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision pointbecause there are two outcomes either true or false.

Static Testing

In this section, we are going to understand **Static testing**, which is used to check the application without executing the code. And we also learn about **static Testing, why we use static Testing, how to perform it, a different technique for static Testing, advantages of static testing, and various Static Testing tools.**

Introduction to Static Testing

Static testing is a verification process used to test the application without implementing the code of the application. And it is a **cost-effective process**.

To avoid the errors, we will execute Static testing in the initial stage of development because it is easier to identify the sources of errors, and it can fix easily.

In other words, we can say that **Static testing** can be done manually or with the help of tools to improve the quality of the application by finding the error at the early stage of development; that is also called the **verification process**.

We can do some of the following important activities while performing static testing:

- **Business requirement review**
- **Design review**
- **Code walkthroughs**
- **The test documentation review**

Note: Static testing is performed in the white box testing phase, where the developer checks every line of the code before giving it to the Test Engineer.

Static testing also helps us to identify those errors which may not be found by Dynamic Testing.

Why do we need to perform Static Testing?

We can perform static testing to fulfill the below aspects:

- We can use static testing to improve the development productivity.
- If we performed static testing on an application, we could find the detects in the earlier stages and easily fix them.
- The usage of static testing will decrease the testing cost, development timescales, and time.

What are the different features we can test in Static Testing?

We can test the various testing activities in Static Testing, which are as follows:

- BRD [Business Requirements Document]
- Functional or system Requirements
- Unit Use Cases
- Prototype
- Prototype Specification Document
- Test Data
- DB Fields Dictionary Spreadsheet
- Documentation/Training Guides/ User Manual
- Test Cases/Test Plan Strategy Document
- Traceability Matrix Document
- Performance Test Scripts/Automation

When we performed Static Testing?

To perform static testing, we need to follow the below steps:

Step1: To review the design of the application entirely, we will perform the **inspection process**.

Step2: After that, we will use a checklist for each document under review to make sure thatall reviews are covered completely.

We can also implement several activities while performing static testing, which are discussed in the following table:

Activities	**Explanation**
Architecture Review	o The architecture review activities contain all business-level processes such as **network diagram, load balancing, server locations, protocol definitions, test equipment, database accessibility, etc.**
Use Cases Requirements Validation	o It is used to authenticates all the end-user actions along with associated input and output. o If the use case is more comprehensive and detailed, we can make more precise and inclusive test cases.
Functional Requirements Validation	o The functional requirement validation activity is used to make sure that all necessary elements identify correctly. o And it also took care of **the software, interface listings, network requirements, hardware, anddatabase functionality.**
Field Dictionary Validation	o In the field dictionary validation, we will test each field in the user interface specified to create field-level validation test cases. o And we can check the fields for **error messages,**

	minimum or maximum length, list values, etc.
Prototype/Screen Mockup Validation	o The prototype validation activity contains the authentication of requirements and uses cases.

Why we need Static Testing?

We required Static testing whenever we encounter the following situation while testing an application or the software:

- **Dynamic Testing is time-consuming**
- **Flaws at earlier stages/identification of Bugs**
- **Dynamic Testing is expensive**
- **Increased size of the**

softwareDynamic Testing is time-

consuming

We need static testing to test the application as dynamic testing is time-taking process even though the dynamic testing identifies the bug and provides some information about the bug.

Flaws at earlier stages/identification of Bugs

When we are developing the software, we cannot completely rely on Dynamic testing as it finds the bugs or defects of the application/software at a later stage because it will take the programmer's plenty of time and effort to fix the bugs.

Dynamic Testing is expensive

We need to perform the static testing on the software product because dynamic testing is more expensive than static testing. Involving the test cases is expensive in dynamic testing as the test cases have been created in the initial stages.

And we also need to preserve the implementation and validation of the test case, whichtakes lots of time from the test engineers.

Increased size of the software

Whenever we test the software, it will increase the size of the software product, which we cannot handle because of the reduction in the productivity of code coverage.

That is why we require static testing to get free from the bugs or defects earlier in thesoftware development life cycle.

Objectives of Static testing

The main objectives of performing static testing is as below:

- Static testing will decrease the flaws in production.
- Static testing will identify, anticipate and fix the bugs at the earliest possible time.
- It is used to save both time and cost.
- It is used to identify defects in the early stage of SDLC, where we can fix them easily.

Some useful points for Successful Static Testing Process

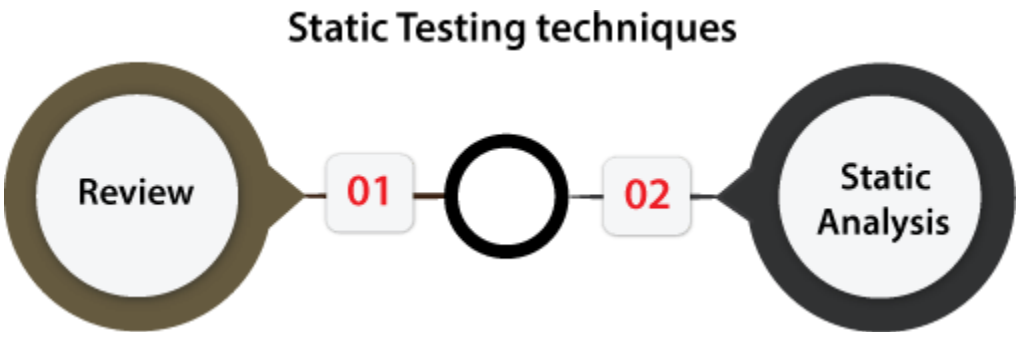
The following guidelines help us to perform a successful static testing process in Software testing.

- We can train participants with examples.
- The testing cost and time can decrease if we delete the major delays in test execution.
- We can retain the process formal as per the project culture.
- We can make emphasis only on things that matter.
- As we know that a software walkthrough and review are usually merged into peer reviews; therefore, we can plan explicitly and track the review activities.
- We can resolve the client's problems.

Static Testing Techniques

Static testing techniques offer a great way to enhance the quality and efficiency of software development. The Static testing technique can be done in two ways, which are as follows:

- **Review**
- **Static Analysis**



Review

In static testing, the **review** is a technique or a process implemented to find the possible bugs in the application. We can easily identify and eliminate faults and defects in the various supporting documents such as SRS **[Software Requirements Specifications] in the review process.**

In other words, we can say that a **review** in Static Testing is that where all the team members will understand about the project's progress.

In static testing, **reviews** can be divided into **four different parts**, which are as follows:

- **Informal reviews**
- **Walkthroughs**
- **Technical/peer review**
- **Inspections**

Let's understand them in detail one by one:

- **Informal reviews**
  In **informal review**, the document designer place the contents in front of viewers, and everyone gives their view; therefore, bugs are acknowledged in the early stage.
- **Walkthrough**
  Generally, the **walkthrough review** is used to performed by a skilled person or expert to verify the bugs. Therefore, there might not be problem in the developmentor testing phase.
- **Peer review**
  In **Peer review**, we can check one another's documents to find and resolve the bugs, which is generally done in a team.
- **Inspection**
  In review, the **inspection** is essentially verifying the document by the higher authority, **for example**, the **verification of SRS [software requirement specifications] document.**

Static Analysis

Another Static Testing technique is **static analysis**, which is used to contain the assessment of the code quality, which is established by developers.

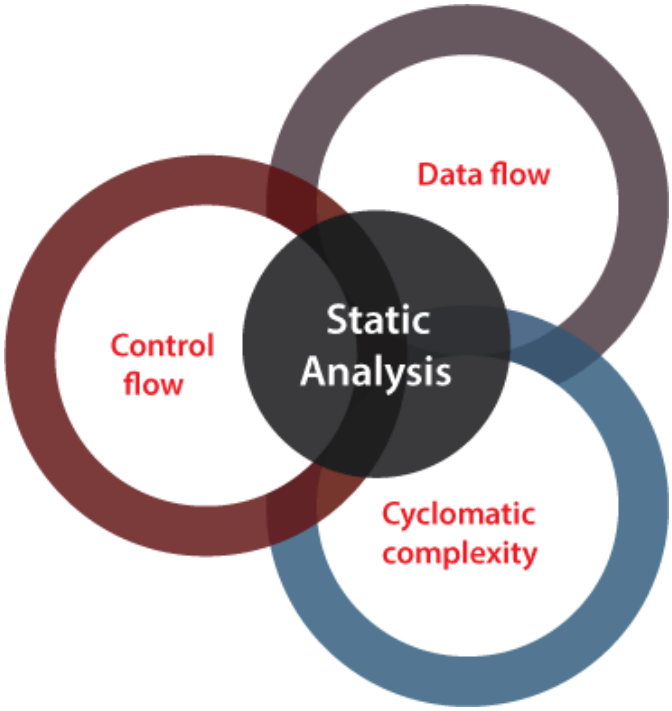
We can use the different tools to perform the code's analysis and evaluation of the same.

In other words, we can say that developers' developed code is analyzed with some tools for structural bugs, which might cause the defects.

The **static analysis** will also help us to identify the below errors:

- **Dead code**
- **Unused variables**
- **Endless loops**
- **Incorrect syntax**
- **Variable with undefined value**

In static testing, **Static Analysis** can be further classified into **three parts**, which are asdiscuss below:



Data Flow: In static analysis, the data flow is connected to the stream processing.

Control Flow: Generally, the control flow is used to specify how the commands orinstructions are implemented.

Cyclomatic Complexity: It is the measurement of the program's complexity, which is mostly linked to the number of independent paths in the control flow graph of the program.

Tools used for Static Testing

In static testing, we have several tools in the market, but here we are discussing the most commonly used tools, which are as follow:

- **CheckStyle**
- **SourceMeter**
- **Soot**

CheckStyle

It is a development tool that is used to help the developers write Java code, which follows a coding standard. The **CheckStyle tool** automates the process of checking Java code.

It is a highly configured tool, which is made to support almost any coding standard. The **Google Java Style, Sun code conventions** are those configuration files, which issupported by CheckStyle.

Feature of CheckStyle

Following are the most common features of CheckStyle:

- It can check various characteristics of our source code.
- The CheckStyle code has the capability to verify the code layout and formatting issues.
- It can also help to identify the method design problems, class design problems.

SourceMeter

It is an advanced tool for the specific static source code analysis of various programming languages such as **C/C++, C#, Java, Python**, and **RPG projects**.

With the **SourceMeter tool's** help, we can easily identify the vulnerable spots of a system under development from the source code.

The free version with partial functionality of SourceMeter can be accessible for all programming languages.

In SourceMeter, we can use the output of the analysis, the quality of the analyzed sourcecode to enhance and developed both the short and long term in a directed way.

Feature of SourceMeter

The most commonly used features of the SourceMeter tool are as follows:

- It provides the most precise coding error detection.
- The SourceMeter tool will provide a deep static code analysis.
- It improved the user interface with the help of third-party integration.
- It will provide platform-independent command-line tools.

Soot

It is a **Java optimization framework**, which means that it is a framework for analyzing and transforming Java and Android applications where we can test the following aspects:

- Named module and modular jar files.
- Automatic modules, which means the modules are repeatedly created from jars in the module-path.
- Exploded modules
- Resolving modules in Soot's

And a Soot can also produce possibly transformed code in the various output formats such as **Android bytecode, Java bytecode Jasmin, and Jimple**.

Advantages of Static Testing

The advantages of static testing are as follows:

- **Improved Product quality**
  Static testing will enhance the product quality because it identifies the flaws or bugsin the initial stage of software development.
- **Improved the efficiency of Dynamic testing**
  The usage of Static testing will improve Dynamic Testing efficiency because the code gets cleaner and better after executing Static Testing.
  As we understood above, static Testing needs some efforts and time to generate and keep good quality test cases.
- **Reduced SDLC cost**
  The Static Testing reduced the SDLC cost because it identifies the bugs in the earlier stages of **the software development life cycle**. So, it needs less hard work and timeto change the product and fix them.

- **Immediate evaluation & feedback**
  The static testing provides us immediate evaluation and feedback of the software during each phase while developing the software product.
- **Exact location of bug is traced**
  When we perform the static testing, we can easily identify the bugs' exact location compared to the dynamic Testing.

Overview

In the Static testing section, we have learned the following topics:

- Static testing is used to identify the faults in the early stage of the Software development cycle [SDLC].
- We have understood that Static Testing is not a replacement for dynamic Testing because both testings identify different bug types.
- We have understood the objective of Static Testing.
- In static testing, the reviews are the productive approach to test the application because the reviews help identify the bugs and recognize the design flaws, missing requirements, and non-maintainable code, etc.
- We have understood several static testing tools, which help us enhance testing performance for the software product.

Dynamic Testing

In this section, we are going to understand Dynamic testing, which is done when the code is executed in the run time environment.

And we also learn about Dynamic testing, why we use it, how to perform it, what are a different technique for Dynamic testing, various tools for Dynamic Testing.

Introduction to Dynamic Testing

Dynamic testing is one of the most important parts of Software testing, which is used to analyse the code's dynamic behavior.

The dynamic testing is working with the software by giving input values and verifying if the output is expected by implementing a specific test case that can be done manually or withan automation process.

The dynamic testing can be done when the code is executed in the run time environment. Itis a validation process where functional testing [unit, integration, system, and user

acceptance testing] and non-functional testing [Performance, usability, compatibility, recovery and security testing] are performed.

As we know that Static testing is a verification process, whereas dynamic testing is a validation process, and together they help us to deliver a cost-effective quality Software product.

Why do we need to perform Dynamic Testing?

We can easily understand how to implement dynamic testing during the STLC [Software Testing Life Cycle] if we consider the characteristics accessible by dynamic testing.

Using dynamic testing, the team can verify the software's critical features, but some of those can be left without any assessment. And they can also affect the functioning, reliability, and performance of the software product.

Hence, we can perform Dynamic testing to fulfill the various below aspects:

- We will perform dynamic testing to check whether the application or software is working fine during and after installing the application without any error.
- We can perform dynamic testing to verify the efficient behavior of the software.
- The software should be compiled and run if we want to perform dynamic testing.
- Generally, Dynamic Testing is implemented to define the dynamic behavior of code.
- The team implements the code to test the software application's performance in a run-time environment during the dynamic testing process.
- It makes sure that the concurrency of the software application with the customer's potentials, needs and the end-user.
- It is an operative technique to measure the effect of several environmental stresseson the software application like network, hardware

Characteristic of Dynamic Testing

For understanding the fundamental of the software testing techniques, we have to learn their attribute and several other components. Hence, following are some of the important characteristics of dynamic testing:

- It is implemented throughout the validation stage of software testing.
- Dynamic Testing is done by performing the program.
- Both functional and non-functional testing include in dynamic testing.
- In Dynamic testing, we can easily identify the bugs for the particular software.
- It helps the team in validating the reliability of the software application.

- Unlike static testing, the team implements the software's code to get expected outputs in dynamic testing.
- Dynamic testing is performed directly on the software application as compare to other testing techniques.
- Dynamic testing is a more formal testing approach for different testing activities such as test execution, coverage consideration, reporting and test case identification.

Dynamic testing Process

Generally, dynamic testing follows a set process when the approach and test implementation performances are decided, and the team can move to execute the differenttesting activities.

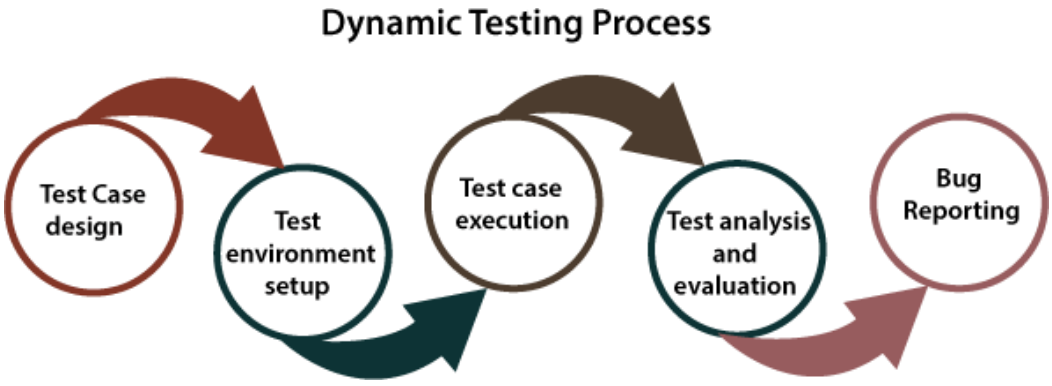
With the help of this process, the team can find any irregularity from the approaches andstrategies and help us display all the testing steps.

In the STLC, the process of Dynamic Testing involves different functions. And all the functions in the dynamic testing process rely on the conclusion of the earlier task in the testing process.

The Dynamic testing process will complete in the following steps:

- Test case design
- Test environment step-up
- Test case execution
- Test analysis and evaluation
- Bug Reporting

The actual Dynamic Testing Process begins from Test Case Design in the software testing lifecycle. Now, we discuss each step one by one to get complete knowledge of the dynamic testing process.



Step1: Test Case Design

In the first step of the dynamic testing process, the teams will design the test cases. Here,we are creating those test cases that depend on the requirements and scope of testing established before the start of the project.

In this step, we can originate the test conditions, obtain the test cases, extract the coverage Items, and identify those features that need to be tested.

Step2: Environment Setup

In the test environment phase, we will make sure that the testing environment should always be parallel to the production environment because the testing is implemented directly on the software product.

In this step, the dynamic testing process's main objective is to install the test environment, which helps us succeed in the test machines.

Step3: Test Execution

Once we successfully install the test environment, we will execute those test cases preparedin the primary stage of the dynamic testing process.

Step4: Analysis & Evaluation

After executing the test cases, we will analyse and evaluate the outcomes derived from the testing. And we will compare those outcomes with the expected results.

If expected and actual results are not the same according to executing, we will consider those test cases as fail, and log the Bug in the bug repository.

Step5: Bug Reporting

After analyzing the test cases, we will be reported and recorded any bugs or defects between the actual result and expected result to the concerned person. And the concernedperson will make sure that the issue has been solved and delivering a quality product.

Example of Dynamic Testing

Let us take one sample example where we understand how dynamic testing will woks.So,

for this, we will understand the login module of any application, such as www. Twitter.com.

Suppose we want to create one new account with a secure password, so we need to follow some pre-defined rules in the password field.

And the password should have eight characters long, capital letters and at least one special character.

If we are testing this functionality, we would take all the input conditions to test this andthen verify the output.

We can also put the non-working constraints, such as input a 4-character password, and validate if there is an error occurred or not.

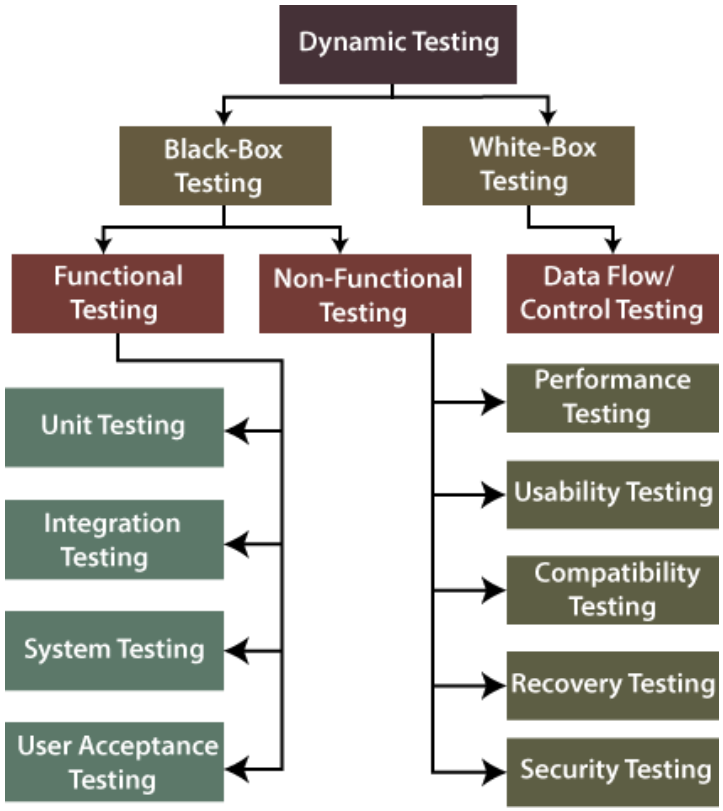
Types of Dynamic testing

Dynamic testing divided into two different testing approach, which are as follows:

- White-box testing
- Black-box testing

Both the testing techniques will help us execute the dynamic testing process efficiently asthey play an important role in verify the performance and quality of the software.

Let's understand them one by one in detail and also see the below diagram of it:



White-box testing

The word white box is used to describe the core perspective of the system. The developerswill perform the white box testing, where they will test every line of the program's code.

When the developers perform the White-box testing and then send the software applicationto the testing team, the testing team will do the black box testing, validate the application as well as the requirements. The white-box testing is further divided into data flow/control testing.

Data flow Testing

The data flow testing is used to identify the program's test paths as per the settings of descriptions and uses of variables in the program. And it does not relate to data flow diagrams.

Black-box testing

The black-box testing is a testing technique where the test engineer selects a module and gives an input value to observe its functionality and analysis of whether the function is giving the expected output or not. If the function produced the correct output, then the particular function will be marked as pass.

To perform black-box testing, the test engineer should have specific knowledge about the software's requirement rather than programming knowledge of the software.

And then, they can develop the test cases to check the correctness of the software's functionality.

Black-box testing is further classified into two types, which are as follows:

- Functional testing
- Non-function testing

Functional testing

Functional testing is one of the most important parts of black-box testing. It mainly focuses on application specification rather than the actual code, and the test engineer will test the program rather than the system.

The functional testing is used to validate the software application's functionality, whether the function is working as per the requirement specification.

In functional testing, each module has been tested by giving the value, determining theoutput, and verifying the actual output with the expected value.

The functional testing is classified into four different type of testing, which are as follows:

- Unit testing
- Integration testing
- System testing

- User acceptance testing

Unit testing

- The unit testing is the first level of functional testing to perform any testing on the software application.
- We will perform the unit testing whenever the application is ready and given to the Test engineer. He/she will start checking every component of the module or application independently or one by one. And this process is known as components testing.
- The primary objective to perform unit testing is to test the correctness of remote code and validate the unit components with their performance.

Integration testing

- When we have successfully done the unit testing on the specific software, we will go for the integration testing. The integration testing will help us to combined individual units and tested as a group. And it is the second levelof functional testing.
- When all the components or modules are working independently, we will check the data flow between the dependent modules, which is known as integration testing.
- The developers and the test engineer perform the integration testing. And the main purpose of the integration is to identify the faults in the interaction between the integrated units.

System testing

- System testing is used to check the end-to-end flow of an application or the software as a user.
- System testing is also known as end-to-end testing as the testing environment is similar to the production environment.
- In the third level (system testing) of functional testing, we go through all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.

User acceptance testing

- The user acceptance testing is performed to certify the system according to requirements. The customer or client does it before accepting the final product.

- In other words, we can say that the UAT is done by the customer (domain expert) for their satisfaction and check whether the application is working according to given business scenarios and real-time scenarios.
- It is the last level of functional testing, which is execute before releasing the software to the market or production environment where two or more end-users will involve.

Non- Functional testing

Another part of black-box testing is non-functional testing. It is used to test non-functional constraints like load test, reliability, performance, and software accountability.

The main objective of performing the non-functional testing is to test the software system's reading speed according to the non-functional parameters because these parameters are never tested before the functional testing.

Non-functional testing plays a vital role in customer satisfaction while testing the software or the application.

It reduces the risk of production and related costs of the software, and it provides athorough knowledge of product behavior and used technologies.

Furthermore, the non-functional testing is divided into various parts, which can be performed at the test level.

- Performance testing
- Usability testing
- Compatibility testing
- Recovery testing
- Security testing

Let's understand them in details one by one:

Performance Testing

- The performance testing is the most importantly used type of non-functional
- Once the software is stable and moved to the production, and it may be accessed by multiple users concurrently, we will do performance testing.
- The performance testing is testing where we check the *behavior of an application by applying some load.*

- As we know it is non-functional testing, which doesn't mean that we always use performance testing when the application is functionally stable; only then we go for performance testing.

Usability Testing

- In usability testing, we will check the user-friendliness, efficiency, and accuracy ofthe software application.
- If we are using usability testing, it ensures that the developed software is easy to test while using the system without facing any problem and makes end-user life easier.

Compatibility testing

- The next type of non-functional testing is compatibility testing, which is used to check the functionality of an application on different software, hardware platforms, network, and browsers.
- The compatibility testing is not performed for all the applications; we will use the compatibility testing only for those applications where we don't have control over the platform used by users.

Recovery testing

- In recovery testing, we can verify how well a system can recover from hardware failures and crashes.
- It reproduced the failure modes or essential producing failures in a controlled environment.
- The recovery testing is performed to confirm that a system is fault-tolerant and can improve well from failures.

Security testing

- The security testing is used to discover the weaknesses, risks, or threats in the software application and help us stop the nasty attack from outsiders and ensure our software applications' security.
- The main purpose of security testing is to identify all the possible uncertainties and vulnerabilities of the application so that the software does not stop working.

Advantages and disadvantages of Dynamic Testing

From detecting and evaluating several bugs and errors in the software to verifying the software's performance, dynamic testing provides serval benefits to the users and the testing team.

However, we have various advantages of dynamic testing as well as some *disadvantages.*

Therefore, below we listed some of the advantages and disadvantages of dynamic testing:

Advantages

Following are the advantages of dynamic testing:

- It validates the performance of the software application.
- The usage of dynamic testing ensures the reliability and constancy of the software product.
- It can automate with the help of tools that detect the problematic and complex bugsin the testing process, which cannot be covered through static Analysis.
- It helps the testing team to identify the weak areas of the run-time environment.
- The most important benefit of using dynamic testing over static testing is the relatively higher number of bugs can be found.
- As compared to static testing, dynamic testing requires a smaller number of meetings at the planning level of testing.
- It implements the software, end to end, and delivers Bug-free software.
- It becomes an essential tool for identifying any security threats.
- In dynamic testing, we can detect the problematic bugs which may have escaped the review processes.
- It also identifying those bugs which cannot be noticed by static testing.
- Dynamic testing can also find security threats, which ensure a better and secure application.

Disadvantages

Following are drawbacks of dynamic testing:

- It is a time-consumingprocess as it implements the software application or code, which needs a massive resource.
- The dynamic testing process is a bit costlieras it increases the budget of the software.

- The dynamic testing needs more human resources to complete the task, which makes its implementation costlier.
- Generally, dynamic testing is executed after the coding phase is completed, and therefore, the bugs are identified later in the life cycle.

Overview

In the dynamic testing section, we have learned the following topics:

- After understood the dynamic testing above, we can easily say that the importanceof dynamic testing is massive in the software testing life cycle (STLC).
- Dynamic testing is used to perform the dynamic behavior of the code.
- We have understood the process of dynamic Testing and the various types of dynamic testing.
- In dynamic testing, we can directly implement the software tests to verify the functional performance, behavior, reliability, and other significant features of the software.
- We have understood the advantages and disadvantages of dynamic testing.

Statement Coverage Testing

Statement coverage is one of the widely used software testing. It comes under white box testing.

Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code out of total statements presentin the source code.

Statement coverage derives scenario of test cases under the white box testing process which is based upon the structure of the code.

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

In white box testing, concentration of the tester is on the working of internal source codeand flow chart or flow graph of the code.

Generally, in the internal source code, there is a wide variety of elements like operators, methods, arrays, looping, control statements, exception handlers, etc. Based on the input

given to the program, some code statements are executed and some may not be executed. The goal of statement coverage technique is to cover all the possible executing statements and path lines in the code.

Let's understand the process of calculating statement coverage by an example:

Here, we are taking source code to create two different scenarios according to input valuesto check the percentage of statement coverage for each scenario.

Source Code Structure:

- Take input of two values like a=0 and b=1.
- Find the sum of these two values.
- If the sum is greater than 0, then print "This is the positive result."
- If the sum is less than 0, then print "This is the negative result."

```
1. input (int a, int b)
2. {
3.  Function to print sum of these integer values (sum = a+b)
4.  If
(sum>0)5. {
6.  Print (This is positive result)
7.  } else
8.  {
9.  Print (This is negative result)
10. }
11. }
```

So, this is the basic structure of the program, and that is the task it is going to do.

Now, let's see the two different scenarios and calculation of the percentage of StatementCoverage for given source code.

Scenario 1:
If a = 5, b = 4

```
1. print (int a, int b) {
2. int sum = a+b;
3. if (sum>0)
4. print ("This is a positive result")
5. else
```

6. print ("This is negative
result")7. }

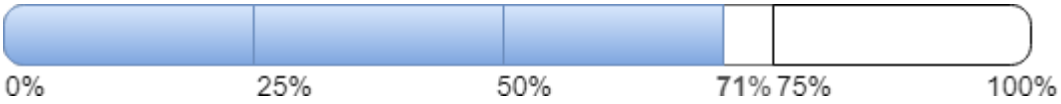
In scenario 1, we can see the value of sum will be 9 that is greater than 0 and as per the condition result will be "This is a positive result." The statements highlighted in yellow color are executed statements of this scenario.

To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 5.

1. Total number of statements = 7
2. Number of executed statements = 5

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

1. Statement coverage = 5/7*100
2. = 500/7
3. = 71%



Likewise, in scenario

2,Scenario 2:
If A = -2, B = -7

1. print (int a, int b) {
2. int sum = a+b;
3. if (sum>0)
4. print ("This is a positive result")
5. else
6. print ("This is negative
result")7. }

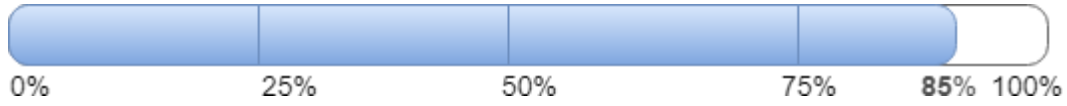
In scenario 2, we can see the value of sum will be -9 that is less than 0 and as per the condition, result will be "This is a negative result." The statements highlighted in yellow color are executed statements of this scenario.

To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 6.

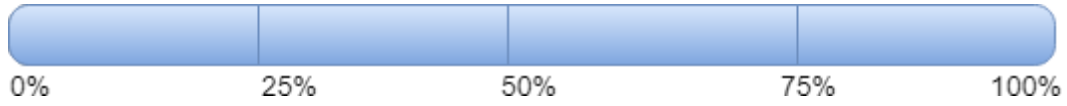
Total number of statements = 7
Number of executed statements = 6

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

1. Statement coverage = 6/7*100
2. = 600/7
3. = 85%



But, we can see all the statements are covered in both scenario and we can consider thatthe overall statement coverage is 100%.



So, the statement coverage technique covers dead code, unused code, and branches.

Decision Coverage Testing

Decision coverage technique comes under white box testing which gives decision coverage to Boolean values. This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.

Generally, a decision point has two decision values one is true, and another is false that's why most of the times the total number of outcomes is two. The percent of decision coverage can be found by dividing the number of exercised outcome with the total numberof outcomes and multiplied by 100.

$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} *100$$

In this technique, it is tough to get 100% coverage because sometimes expressions get complicated. Due to this, there are several different methods to report decision coverage. All these methods cover the most important combinations and very much similar to decision coverage. The benefit of these methods is enhancement of the sensitivity of control flow.

We can find the number of decision coverage as follows.

Let's understand it by an example.

Consider the code to apply on decision coverage technique:

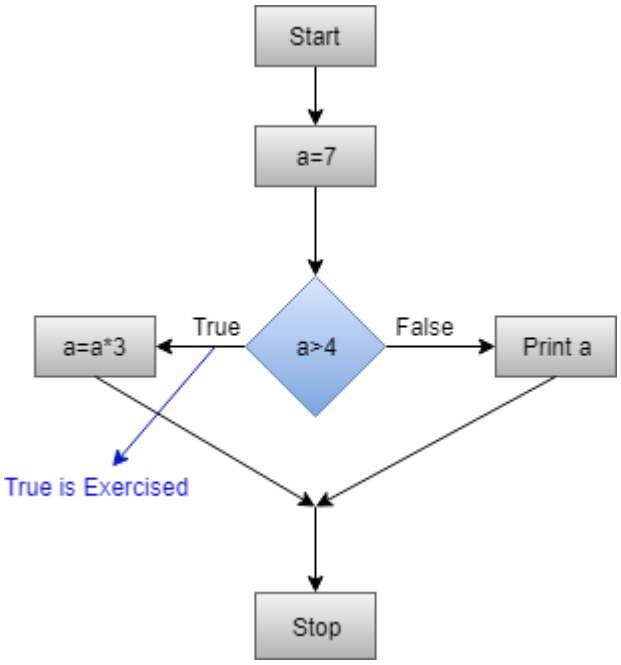
```
1. Test (int a)
2. {
3.   If(a>4)
4.   a=a*3
5. Print (a)
6. }
```

Scenario 1:
Value of a is 7 (a=7)

```
1. Test (int a=7)
2. { if (a>4)
3.  a=a*3
4. print (a)
5. }
```

The code highlighted in yellow is executed code. The outcome of this code is "True" if condition (a>4) is checked.

Control flow graph when the value of a is 7.



Calculation of Decision Coverage percent:

$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} * 100$$

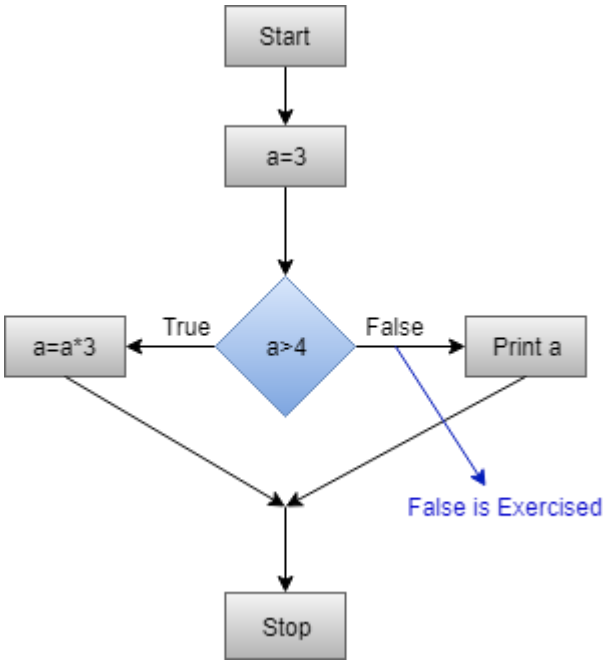
1. Decision Coverage = ½*100 (Only "True" is exercised)2. =100/2
3. = 50
4. Decision Coverage is 50%

Scenario 2:
Value of a is 3 (a=3)

1. Test (int a=3)
2. { if (a>4)
3. a=a*3
4. print (a)
5. }

The code highlighted in yellow will be executed. The outcome of this code is ?False? if condition (a>4) is checked.

Control flow graph when the value of a is 3



Calculation of Decision Coverage percent:

$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} * 100$$

1. = ½*100 (Only "False" is exercised)
2. =100/2
3. = 50
4. Decision Coverage = 50%

Result table of Decision Coverage:

Test Case	Value of A	Output	Decision Coverage
1	3	3	50%
2	7	21	50%

What is Basis Path Testing?

Basis path testing, a structured testing or white box testing technique used for designing test cases intended to examine all possible paths of execution at least once. Creating and

executing tests for all possible paths results in 100% statement coverage and 100% branch coverage.

Example:

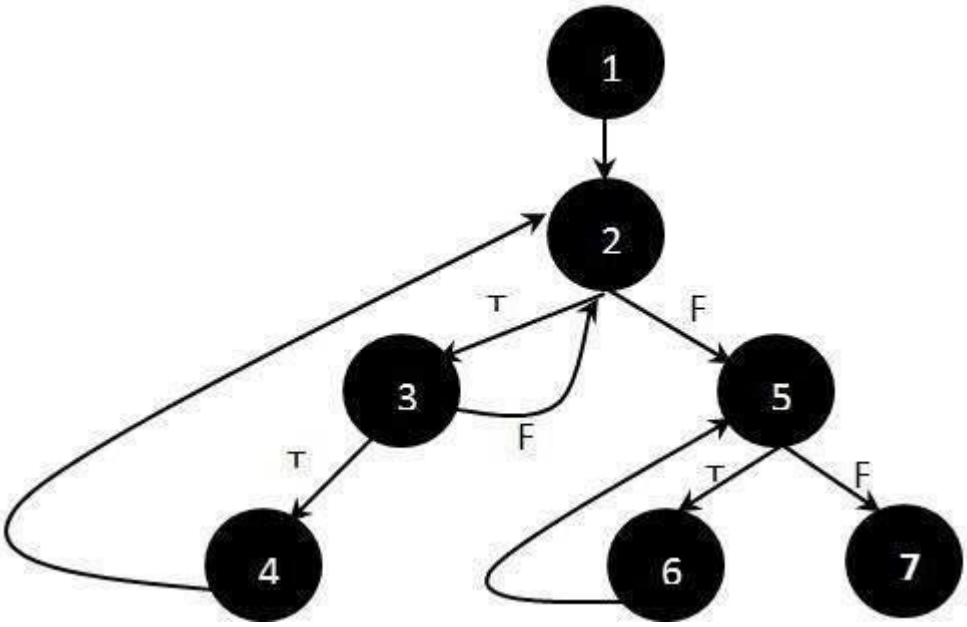
```
Function fn_delete_element (int value, int array_size, int array[])
{
        1 int i;
        location = array_size + 1;

        2 for i = 1 to array_size
        3 if ( array[i] == value )
        4 location = i;
         end    if;
         end for;

        5 for i = location to array_size
        6 array[i] = array[i+1];
        end for;
        7 array_size --;
}
```

Steps to Calculate the independent paths

Step 1 : Draw the Flow Graph of the Function/Program under consideration as shownbelow:



Step 2 : Determine the independent paths.

Path 1: 1 - 2 - 5 - 7

Path 2: 1 - 2 - 5 - 6 - 7
Path 3: 1 - 2 - 3 - 2 - 5 - 6 - 7
Path 4: 1 - 2 - 3 - 4 - 2 - 5 - 6 - 7

Control Flow Testing

Control flow testing is a testing technique that comes under white box testing. The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selectedby the tester to set the testing path. It is mostly used in unit testing. Test cases represented by the control graph of the program.

Control Flow Graph is formed from the node, edge, decision node, junction node to specify all possible execution path.

Notations used for Control Flow Graph

1. Node
2. Edge
3. Decision Node
4. Junction node

Node

Nodes in the control flow graph are used to create a path of procedures. Basically, it represents the sequence of procedures which procedure is next to come so, the tester can determine the sequence of occurrence of procedures.

We can see below in example the first node represent the start procedure and the next procedure is to assign the value of n after assigning the value there is decision node to decide next node of procedure as per the value of n if it is 18 or more than 18 so Eligible procedure will execute otherwise if it is less than 18 Not Eligible procedure executes. Thenext node is the junction node, and the last node is stop node to stop the procedure.

Edge

Edge in control flow graph is used to link the direction of nodes.

We can see below in example all arrows are used to link the nodes in an appropriatedirection.

Decision node

Decision node in the control flow graph is used to decide next node of procedure as per the value.

We can see below in example decision node decide next node of procedure as per the valueof n if it is 18 or more than 18 so Eligible procedure will execute otherwise if it is less than 18, Not Eligible procedure executes.

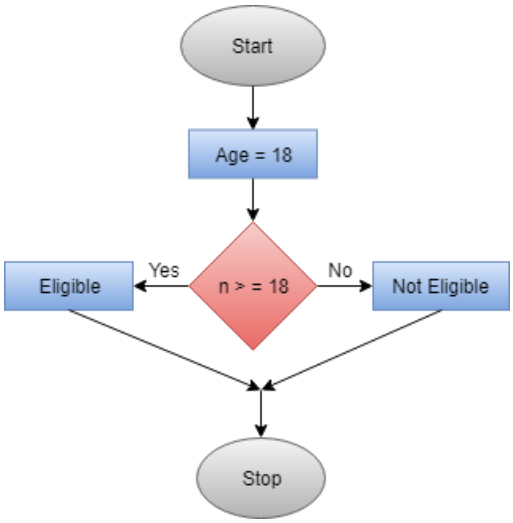
Junction node

Junction node in control flow graph is the point where at least three links meet.

Example

```
1. public class
VoteEligiblityAge{2.
3.  public static void main(String []args){
4.  int
n=45;       5.
if(n>=18)  6.
{
7.      System.out.println("You are eligible for voting");
8.   } else
9.  {
10.   System.out.println("You are not eligible for voting");
11. }
12. }
13. }
```

Diagram - control flow graph



The above example shows eligibility criteria of age for voting where if age is 18 or more than 18 so print message "You are eligible for voting" if it is less than 18 then print "You are not eligible for voting."

Program for this scenario is written above, and the control flow graph is designed for the testing purpose.

In the control flow graph, start, age, eligible, not eligible and stop are the nodes, n>=18 is a decision node to decide which part (if or else) will execute as per the given value. Connectivity of the eligible node and not eligible node is there on the stop node.

Test cases are designed through the flow graph of the programs to determine the execution path is correct or not. All nodes, junction, edges, and decision are the essential parts to design test cases.

Cyclomatic Complexity

Cyclomatic complexity is a software metric used to measure the complexity of a program. Thomas J. McCabe developed this metric in 1976.McCabe interprets a computer program asa set of a strongly connected directed graph. Nodes represent parts of the source code having no branches and arcs represent possible control flow transfers during program execution. The notion of program graph has been used for this measure, and it is used to measure and control the number of paths through a program. The complexity of a computerprogram can be correlated with the topological complexity of a graph.

How to Calculate Cyclomatic Complexity?

McCabe proposed the cyclomatic number, V (G) of graph theory as an indicator of software complexity. The cyclomatic number is equal to the number of linearly independent paths through a program in its graphs representation. For a program control graph G, cyclomatic number, V (G), is given as:

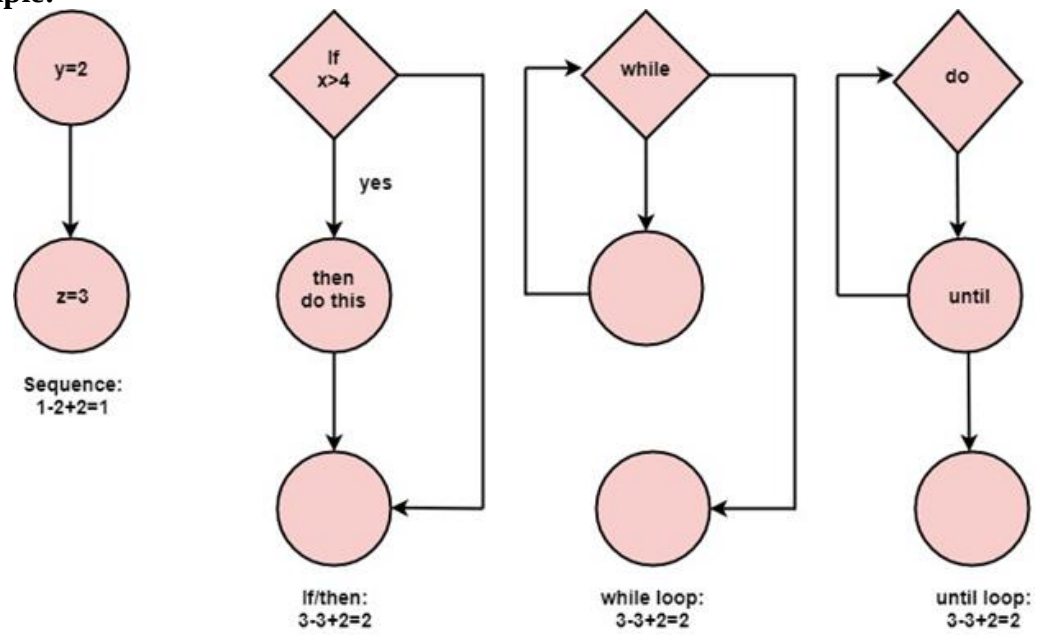
$$V(G) = E - N + 2 * P$$

E = The number of edges in graphs G

N = The number of nodes in graphs G

P = The number of connected components in graph G.

Example:



Properties of Cyclomatic complexity:

Following are the properties of Cyclomatic complexity:

1. V (G) is the maximum number of independent paths in the graph
2. V (G) >=1
3. G will have one path if V (G) = 1
4. Minimize complexity to 10

Mutation Testing

What is mutation testing?

Mutation testing is a white box method in software testing where we insert errors purposely into a program (under test) to verify whether the existing test case can detect the error or not. In this testing, the mutant of the program is created by making some modifications to the original program.

The primary objective of mutation testing is to check whether each mutant created an output, which means that it is different from the output of the original program. We will make slight modifications in the mutant program because if we change it on a massive scalethan it will affect the overall plan.

When we detected the number of errors, it implies that either the program is correct or the test case is inefficient to identify the fault.

Mutation testing purposes is to evaluate the quality of the case that should be able to failthe mutant code hence this method is also known as Fault-based testing as it used to produce an error in the program and that why we can say that the mutation testing is performed to check the efficiency of the test cases.

What is mutation?

The mutation is a small modification in a program; these minor modifications are planned to typical low-level errors which are happened at the time of coding process.

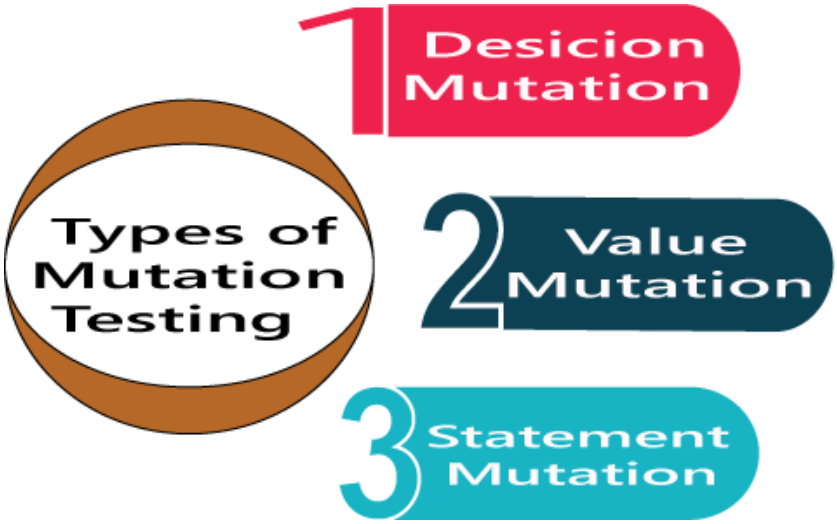
Generally, we deliberate the mutation operators in the form of rules which match the dataand also generate some efficient environment to produce the mutant.

Types of mutation testing

Mutation testing can be classified into three parts, which are as follows:

- Decision mutations
- value mutations
- Statement mutations

Let us understand them one by one:



Decision mutations

In this type of mutation testing, we will check the design errors. And here, we will do the modification in arithmetic and logical operator to detect the errors in the program.

Like if we do the following changes in arithmetic operators:

- plus(+)→ minus(-)

- asterisk(*)→ double asterisk(**)
- plus(+)→incremental operator(i++)

Like if we do the following changes in logical operators

- Exchange P > → P<, OR P>=

Now, let see one example for our better understanding:

Original Code	Modified Code

if(p >q)
r= 5;
else
r= 15;
	if(p < q)
	r = 5;
	else
	r = 15;

Value mutations

In this, the values will modify to identify the errors in the program, and generally, we will change the following:

- Small value à higher value
- Higher value àSmall value.

For Example:

Original Code	Modified Code

int add =9000008;
int p = 65432;
int q =12345;
int r = (p+ q);
	int mod = 9008;
	int p = 65432;
	int q =12345;
	int r= (p + q);

Statement Mutations

Statement mutations means that we can do the modifications into the statements byremoving or replacing the line as we see in the below example:

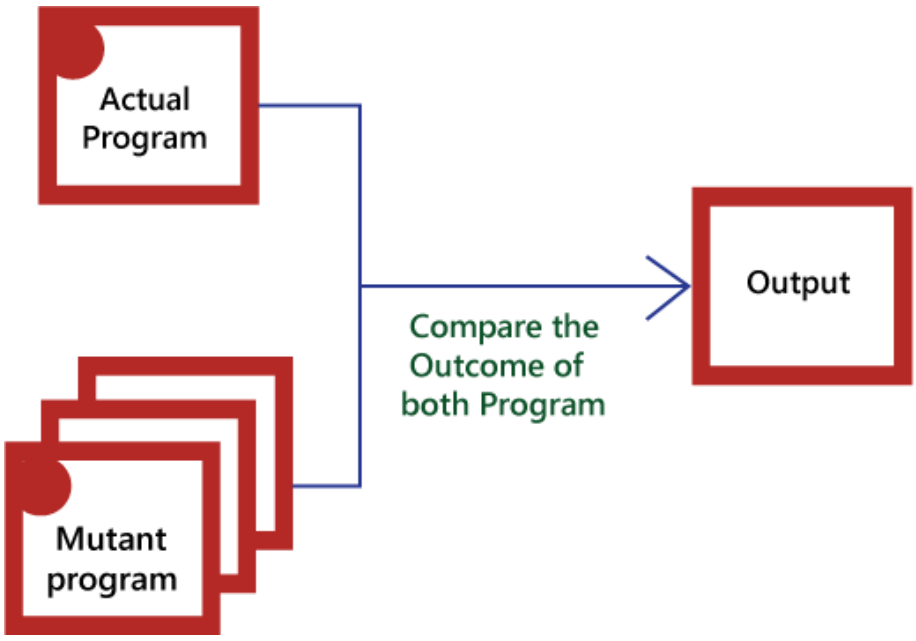
Original Code	Modified Code

if(p * q)
r = 15;
else
r = 25;
	if(p* q)
	s = 15;
	else
	s = 25;

In the above case, we have replaced the statement r=15 by s=15, and r=25 by s=25.

How to perform mutation testing

To perform mutation testing, we will follow the below process:



- In this, firstly, we will add the errors into the source code of the program by producing various versions, which are known mutants. Here every mutant having the one error, which leads the mutant kinds unsuccessful and also validates the efficiency of the test cases.
- After that, we will take the help of the test cases in the mutant program and the actual application will find the errors in the code.
- Once we identify the faults, we will match the output of the actual code and mutant code.
- After comparing the output of both actual and mutant programs, if the results are not matched, then the mutant is executed by the test cases. Therefore the test case

has to be sufficient for identifying the modification between the actual program and the mutant program.

- And if the actual program and the mutant program produced the exact result, then the mutant is saved. And those cases are more active test cases because it helps usto execute all the mutants.

Advantages and disadvantages of Mutation Testing

Advantages

The benefits of mutation testing are as follows:

- It is a right approach for error detection to the application programmer
- The mutation testing is an excellent method to achieve the extensive coverage of the source program.
- Mutation testing helps us to give the most established and dependable structure for the clients.
- This technique can identify all the errors in the program and also helps us to discover the doubts in the code.

Disadvantages

The drawbacks of mutant testing are as follows:

- This testing is a bit of time taking and costlier process because we have many mutant programs that need to be created.
- The mutation testing is not appropriate for Black-box testing as it includes the modification in the source code.
- Every mutation will have the same number of test cases as compare to the actual program. Therefore the significant number of the mutant program may need to be tested beside the real test suite.
- As it is a tedious process, so we can say that this testing requires the automation tools to test the application.

Black box testing

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of blackbox testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the resultto the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Generic steps of black box testing

- The black box test is based on the specification of requirements, so it is examined in the beginning.
- In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- The fourth phase includes the execution of all test cases.
- In the fifth step, the tester compares the expected output against the actual output.
- In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Test procedure

The test procedure of black box testing is a kind of process in which the tester has specific knowledge about the software's work, and it develops test cases to check the accuracy of the software's functionality.

It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function.A tester knows about the definite output of a particular input, but not about how the result is arising. There are various techniques used in black box testing for testing like decision table technique, boundary value analysis technique, state transition, All-pair testing, cause-effect graph technique, equivalence partitioning technique, error guessing technique, use case technique and user story technique. All these techniques have been explained in detail within the tutorial.

Test cases

Test cases are created considering the specification of the requirements. These test cases are generally created from working descriptions of the software including requirements, design parameters, and other specifications. For the testing, the test designer selects both positive test scenario by taking valid input values and adverse test scenario by taking invalidinput values to determine the correct output. Test cases are mainly designed for functional testing but can also be used for non-functional testing. Test cases are designed by the testing team, there is not any involvement of the development team of software.

Techniques Used in Black Box Testing

Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.
Boundary Value Technique	Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limitof a variable. It tests, while entering boundary value whether the software is producing correct output or not.
State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to access the application.
All-pair Testing Technique	All-pair testing Technique is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.
Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result.It is based on a collection of requirements.
Equivalence Partitioning	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it

Technique	is mandatory that all partitions must exhibit the same behavior.
Error Guessing Technique	Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
Use Case Technique	Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

Boundary Value Analysis

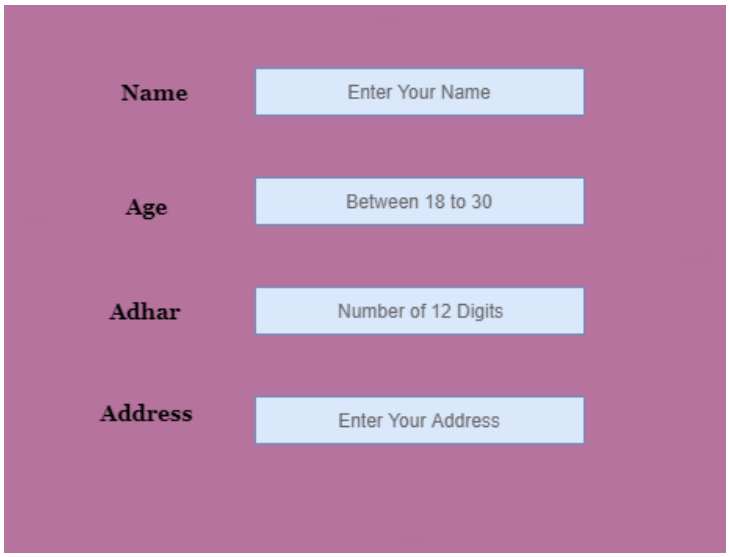
Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

Whenever we do the testing by boundary value analysis, the tester focuses on, whileentering boundary value whether the software is producing correct output or not.

Boundary values are those that contain the upper and lower limit of a variable. Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.

The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error.

There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.



Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential.

Let's understand via practical:

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39

The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

If the software which is under test, follows all the testing guidelines and specifications thenit is sent to the releasing team otherwise to the development team to fix the defects.

Equivalence Partitioning Technique

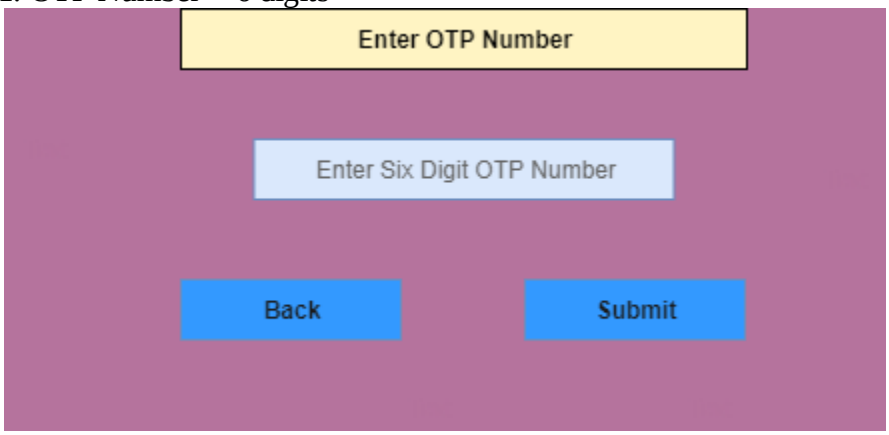
Equivalence partitioning is a technique of software testing in which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior. If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false. The principle of equivalence partitioning is, test cases should be designed to cover each partition at least once. Each value of every equal partition must exhibit the same behavior as other.

The equivalence partitions are derived from requirements and specifications of the software. The advantage of this approach is, it helps to reduce the time of testing due to a smaller number of test cases from infinite to finite. It is applicable at all levels of the testingprocess.

Examples of Equivalence Partitioning technique

Assume that there is a function of a software application that accepts a particular number of digits, not greater and less than that particular number. For example, an OTP number which contains only six digits, less or more than six digits will not be accepted, and the application will redirect the user to the error page.

1. 1. OTP Number = 6 digits

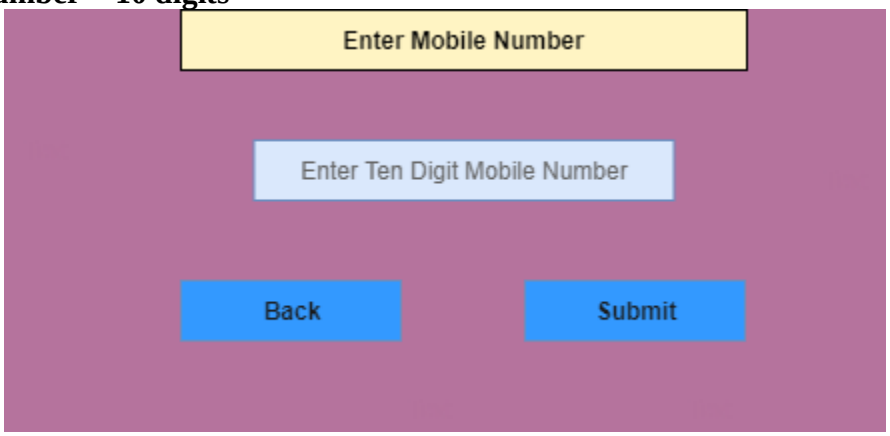


INVALID	INVALID	VALID	VALID
1 Test case	2 Test case	3 Test case	
DIGITS >=7	DIGITS<=5	DIGITS = 6	DIGITS = 6
93847262	9845	456234	451483

Let's see one more example.

A function of the software application accepts a 10 digit mobile number.

1. **2. Mobile number = 10 digits**



INVALID 1 Test case	**INVALID** 2 Test case	**VALID** 3 Test case	**VALID**
DIGITS >=11	DIGITS<=9	DIGITS = 10	DIGITS =10
93847262219	984543985	9991456234	9893451483

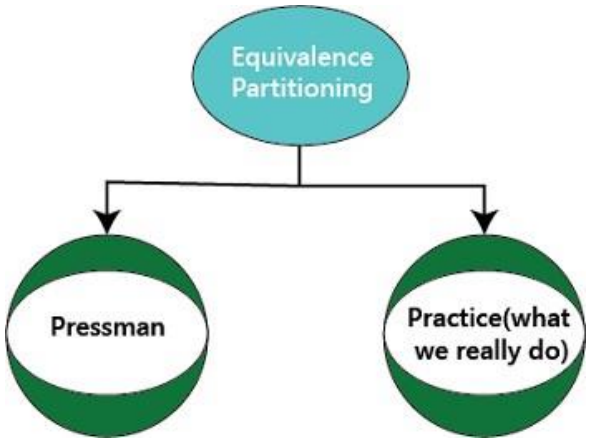
In both examples, we can see that there is a partition of two equally valid and invalid partitions, on applying valid value such as OTP of six digits in the first example and mobile number of 10 digits in the second example, both valid partitions behave same, i.e. redirected to the next page.

Another two partitions contain invalid values such as 5 or less than 5 and 7 or more than 7 digits in the first example and 9 or less than 9 and 11 or more than 11 digits in the second example, and on applying these invalid values, both invalid partitions behave same, i.e. redirected to the error page.

We can see in the example, there are only three test cases for each example and that is alsothe principal of equivalence partitioning which states that this method intended to reduce the number of test cases.

How we perform equivalence partitioning

We can perform equivalence partitioning in two ways which are as follows:



Let us see how pressman and general practice approaches are going to use in differentconditions:

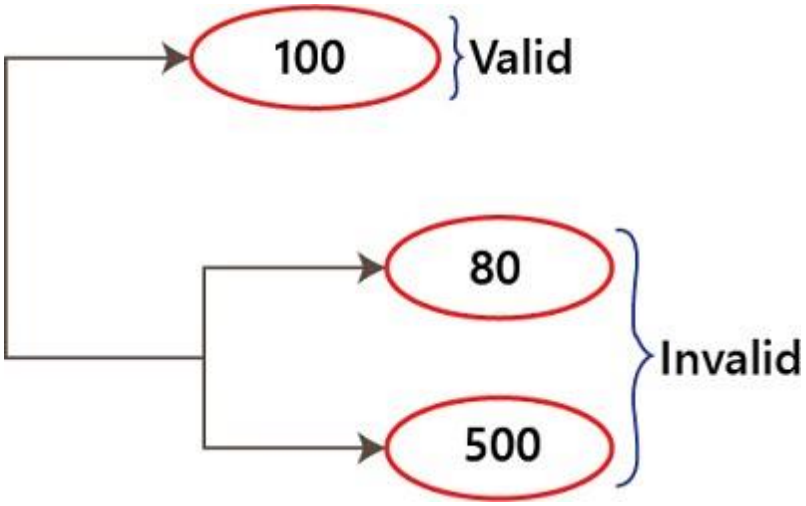
Condition1

If the requirement is **a range of values**, then derive the test case for **one valid** and **two invalid** inputs.

Here, the **Range of values** implies that whenever we want to identify the range values, we go for equivalence partitioning to achieve the minimum test coverage. And after that, we go for error guessing to achieve maximum test coverage.

According to pressman:

For example, the Amount of test field accepts a Range (100-400) of values:

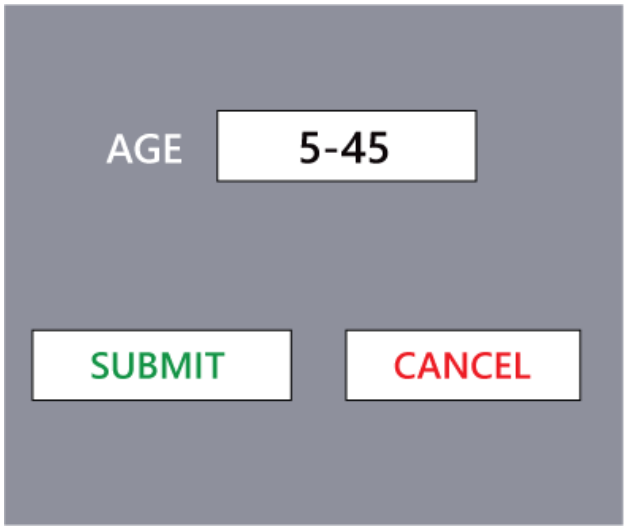


According to General Practice method:

Whenever the requirement is Range + criteria, then divide the Range into the internals and check for all these values.

For example:

In the below image, the pressman technique is enough to test for an age text field for one valid and two invalids. But, if we have the condition for insurance of ten years and above are required and multiple policies for various age groups in the age text field, then we need to use the practice method.



Condition2

If the requirement is **a set of values**, then derive the test case for **one valid** and **two invalid** inputs.

Here, **Set of values** implies that whenever we have to test a set of values, we go for **one positive** and **two negative** inputs, then we moved for error guessing, and we also need to verify that all the sets of values are as per the requirement.

Example 1

Based on the Pressman Method

If the Amount Transfer is (100000-

700000)Then for, 1 lakh →Accept

And according to General Practice method

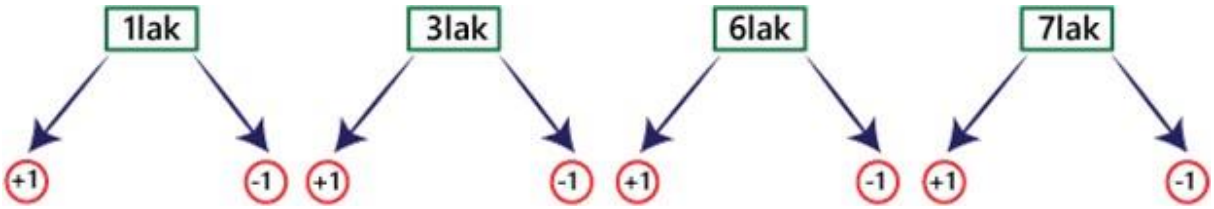
The Range + Percentage given to 1 lakh - 7

lakh**Like:** 1lak - 3lak →5.60%

3lak - 6lak

→3.66%6lak -

7lak →Free



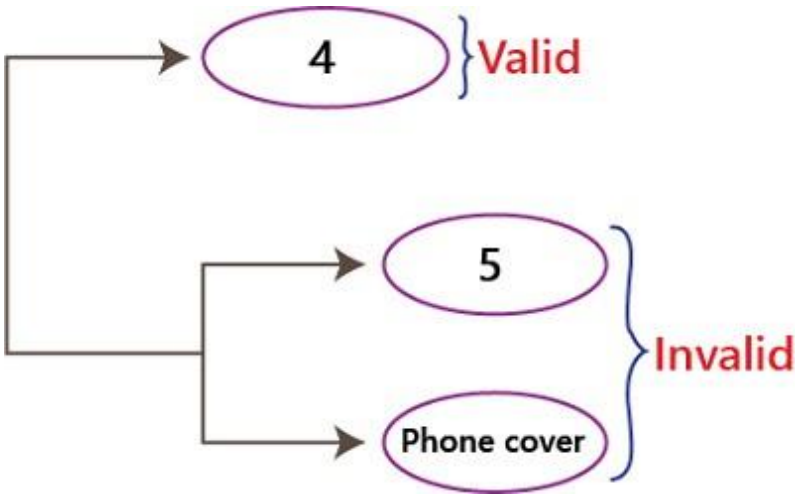
If we have things like loans, we should go for the general practice approach and separate the stuff into the intervals to achieve the minimum test coverage.

Example 2

if we are doing online shopping, mobile phone product, and the different **Product ID - 1,4,7,9**

Here, **1 → phone covers 4 → earphones 7 → charger 9 → Screen guard**

And if we give the product id as **4**, it will be accepted, and it is one valid value, and if we provide the product id as **5 and phone cover**, it will not be accepted as per the requirement, and these are the two invalid values.

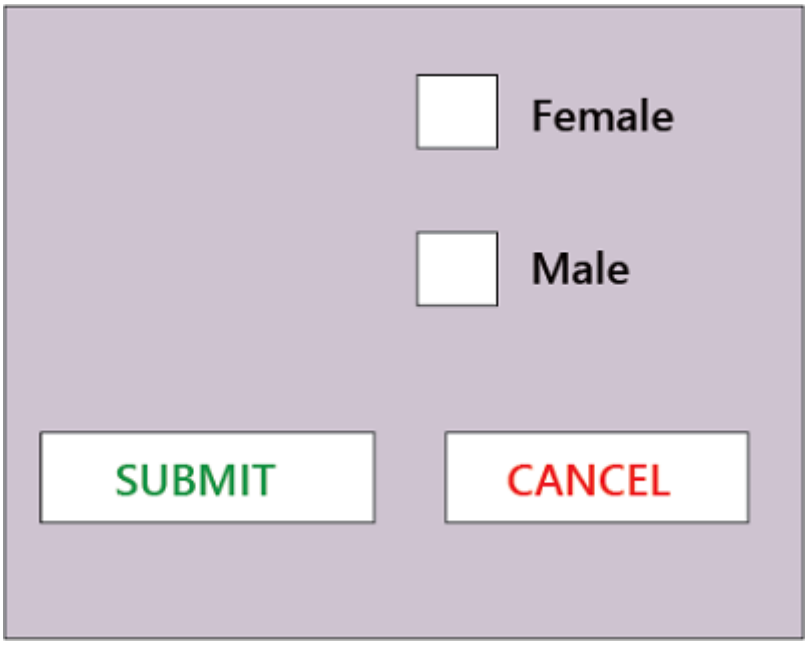


Condition 3

If the requirement id **Boolean (true/false)**, then derive the test case for both true/falsevalues.

The Boolean value can be true and false for the radio button, checkboxes.

For example



Serial no	Description	Input	Expected	Note
1	Select valid	NA	True	---
2	Select invalid	NA	False	Values can be change based according to the requirement.
3	Do not select	NA	Do not select anything, error message should be displayed	We cannot go for next question

4	Select both	NA	We can select any radio button	Only one radio button can be selected at a time.

Note:

In **Practice** method, we will follow the below process:

Here, we are testing the application by deriving the below inputs values:

80	500	1000	2000	3000	4000	5000	6000	7000

Let us see one program for our better understanding.

1. If(amount < **500** or > 7000)
2. {
3. Error Message
4. }
5. if(amount is between 500 & 3000)
6. {
7. deduct 2%
8. }
9. if (amount > 3000)
10. {
11. deduct 3%
12. }

When the pressman technique is used, the first two conditions are tested, but if we use thepractice method, all three conditions are covered.

We don't need to use the practice approach for all applications. Sometime we will use the pressman method also.

But, if the application has much precision, then we go for the practice method.If

we want to use the practice method, it should follow the below aspects:

- It should be product-specific
- It should be case-specific

- The number of divisions depends on the precision(2% and 3 % deduction)

Advantages and disadvantages of Equivalence Partitioning technique

Following are pros and cons of equivalence partitioning technique:

Advantages	Disadvantages
It is process-oriented	All necessary inputs may not cover.
We can achieve the Minimum test coverage	This technique will not consider the condition for boundary value analysis.
It helps to decrease the general test execution time and also reduce the set of test data.	The test engineer might assume that the outputfor all data set is right, which leads to the problem during the testing process.

Cause and Effect Graph in Black box Testing

Cause-effect graph comes under the black box testing technique which underlines the relationship between a given result and all the factors affecting the result. It is used to writedynamic test cases.

The dynamic test cases are used when code works dynamically based on user input. For example, while using email account, on entering valid email, the system accepts it but, when you enter invalid email, it throws an error message. In this technique, the input conditions are assigned with causes and the result of these input conditions with effects.

Cause-Effect graph technique is based on a collection of requirements and used to determine minimum possible test cases which can cover a maximum test area of thesoftware.

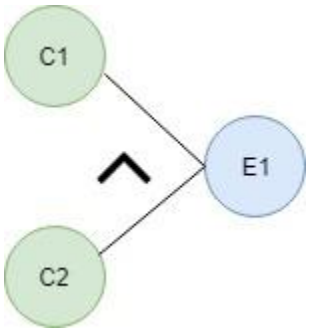
The main advantage of cause-effect graph testing is, it reduces the time of test execution and cost.

This technique aims to reduce the number of test cases but still covers all necessary test cases with maximum coverage to achieve the desired application quality.

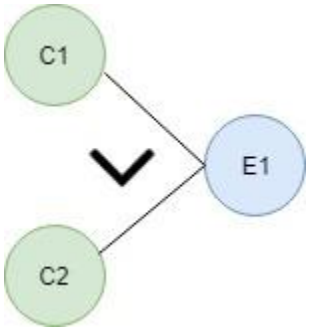
Cause-Effect graph technique converts the requirements specification into a logical relationship between the input and output conditions by using logical operators like AND, OR and NOT.

Notations used in the Cause-Effect Graph

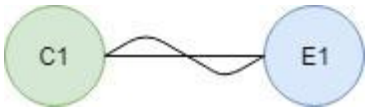
AND - E1 is an effect and C1 and C2 are the causes. If both C1 and C2 are true, then effect E1will be true.



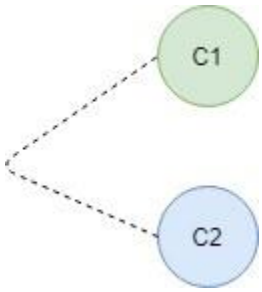
OR - If any cause from C1 and C2 is true, then effect E1 will be true.



NOT - If cause C1 is false, then effect E1 will be true.



Mutually Exclusive - When only one cause is true.



Let's try to understand this technique with some examples:

Situation:

The character in column 1 should be either A or B and in the column 2 should be a digit. If both columns contain appropriate values then update is made. If the input of column 1 is incorrect, i.e. neither A nor B, then message X will be displayed. If the input in column 2 isincorrect, i.e. input is not a digit, then message Y will be displayed.

- A file must be updated, if the character in the first column is either "A" or "B" and in the second column it should be a digit.
- If the value in the first column is incorrect (the character is neither A nor B) then massage X will be displayed.
- If the value in the second column is incorrect (the character is not a digit) then massage Y will be displayed.

Column 1	***Column 2***
Correct value - A or B	*Correct value- Any digit*
Incorrect value - Any character except A or B	*Incorrect value- Any character except digit*

Now, we are going to make a Cause-Effect graph for the above situation:

Causes are:

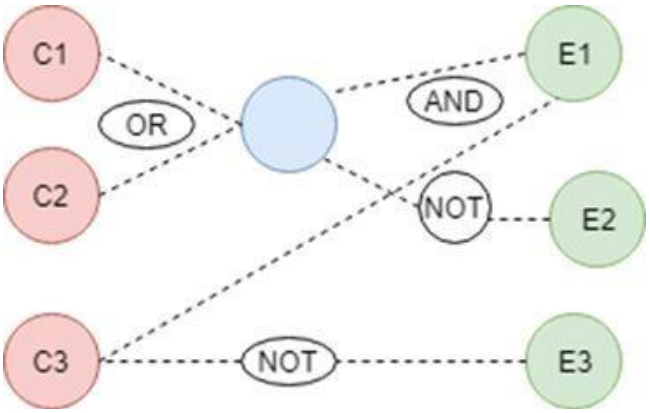
- C1 - Character in column 1 is A
- C2 - Character in column 1 is B
- C3 - Character in column 2 is digit!

Effects:

- E1 - Update made (C1 OR C2) AND C3
- E2 - Displays Massage X (NOT C1 AND NOT C2)
- E3 - Displays Massage Y (NOT

C3) Where AND, OR, NOT are the logical

gates.



Effect E1- Update made- The logic for the existence of effect E1 is "**(C1 OR C2) AND C3**". For **C1 OR C2**, any one from C1 and C2 should be true. For logic **AND C3** (Character in column 2 should be a digit), C3 must be true. In other words, for the existence of effect E1 (Update made) any one from C1 and C2 but the C3 must be true. We can see in graph causeC1 and C2 are connected through OR logic and effect E1 is connected with AND logic.

Effect E2 - Displays Massage X - The logic for the existence of effect E2 is "**NOT C1 AND NOTC2**" that means both C1 (Character in column 1 should be A) and C2 (Character in column 1 should be B) should be false. In other words, for the existence of effect E2 the character in column 1 should not be either A or B. We can see in the graph, **C1 OR C2** is connected through NOT logic with effect E2.

Effect E3 - Displays Massage Y- The logic for the existence of effect E3 is "**NOT C3**" that means cause C3 (Character in column 2 is a digit) should be false. In other words, for theexistence of effect E3, the character in column 2 should not be a digit. We can see in thegraph, **C3** is connected through NOT logic with effect E3.

So, it is the cause-effect graph for the given situation. A tester needs to convert causes and effects into logical statements and then design cause-effect graph. If function gives output (effect) according to the input (cause) so, it is considered as defect free, and if not doing so, then it is sent to the development team for the correction.

Conclusion

Summary of the steps:

- Draw the circles for effects and Causes.
- Start from effect and then pick up what is the cause of this effect.
- Draw mutually exclusive causes (exclusive causes which are directly connected via one effect and one cause) at last.
- Use logic gates to draw dynamic test cases.

Decision table technique in Black box testing

Decision table technique is one of the widely used case design techniques for black box testing. This is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form.

That's why it is also known as a cause-effect table. This technique is used to pick the test cases in a systematic manner; it saves the testing time and gives good coverage to the testing area of the software application.

Decision table technique is appropriate for the functions that have a logical relationship between two and more than two inputs.

This technique is related to the correct combination of inputs and determines the result of various combinations of input. To design the test cases by decision table technique, we needto consider conditions as input and actions as output.

Let's understand it by an example:

Most of us use an email account, and when you want to use an email account, for this you need to enter the email and its associated password.

If both email and password are correctly matched, the user will be directed to the email account's homepage; otherwise, it will come back to the login page with an error message specified with "Incorrect Email" or "Incorrect Password."

Now, let's see how a decision table is created for the login function in which we can log in by using email and password. Both the email and the password are the conditions, and the expected result is action.

Email (condition1)	T	T	F	F
Password (condition2)	T	F	T	F
Expected Result (Action)	Account Page	Incorrect password	Incorrect email	Incorrect email

In the table, there are four conditions or test cases to test the login function. In the first condition if both email and password are correct, then the user should be directed to account's Homepage.

In the second condition if the email is correct, but the password is incorrect then the function should display Incorrect Password. In the third condition if the email is incorrect, but the password is correct, then it should display Incorrect Email.

Now, in fourth and last condition both email and password are incorrect then the function should display Incorrect Email.

In this example, all possible conditions or test cases have been included, and in the same way, the testing team also includes all possible test cases so that upcoming bugs can be cured at testing level.

In order to find the number of all possible conditions, tester uses 2^n formula where n denotes the number of inputs; in the example there is the number of inputs is 2 (one is trueand second is false).

Number of possible conditions = 2^ Number of Values of the second condition
Number of possible conditions =2^2 = 4

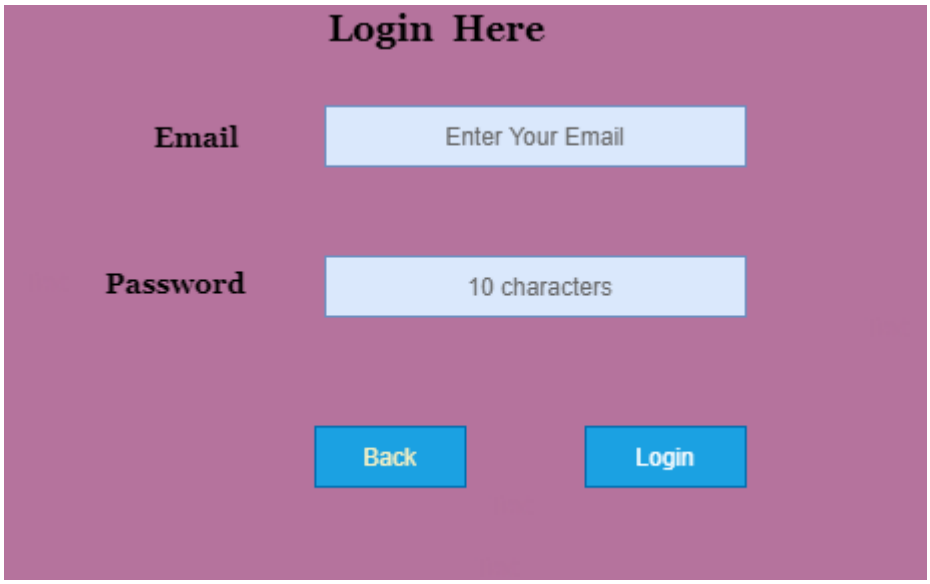
While using the decision table technique, a tester determines the expected output, if the function produces expected output, then it is passed in testing, and if not then it is failed. Failed software is sent back to the development team to fix the defect.

State Transition Technique

The general meaning of state transition is, different forms of the same situation, and according to the meaning, the state transition method does the same. It is used to capturethe behavior of the software application when different input values are given to the same function.

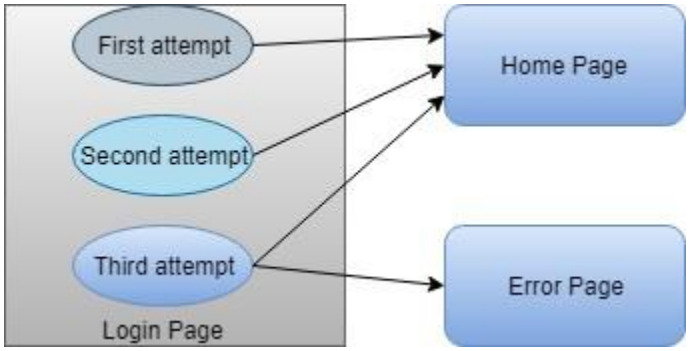
We all use the ATMs, when we withdraw money from it, it displays account details at last. Now we again do another transaction, then it again displays account details, but the details displayed after the second transaction are different from the first transaction, but both details are displayed by using the same function of the ATM. So the same function was usedhere but each time the output was different, this is called state transition. In the case of testing of a software application, this method tests whether the function is following state transition specifications on entering different inputs.

This applies to those types of applications that provide the specific number of attempts to access the application such as the login function of an application which gets locked after the specified number of incorrect attempts. Let's see in detail, in the login function we use email and password, it gives a specific number of attempts to access the application, after crossing the maximum number of attempts it gets locked with an error message.



Let see in the diagram:

There is a login function of an application which provides a maximum three number of attempts, and after exceeding three attempts, it will be directed to an error page.



State transition table

STATE	LOGIN	VALIDATION	REDIRECTED
S1	First Attempt	Invalid	S2
S2	Second Attempt	Invalid	S3

S3	Third Attempt	Invalid	S5
S4	Home Page		
S5	Error Page		

In the above state transition table, we see that state S1 denotes first login attempt. When the first attempt is invalid, the user will be directed to the second attempt (state S2). If the second attempt is also invalid, then the user will be directed to the third attempt (state S3). Now if the third and last attempt is invalid, then the user will be directed to the error page (state S5).

But if the third attempt is valid, then it will be directed to the homepage (state S4).

Let's see state transition table if third attempt is valid:

STATE	LOGIN	VALIDATION	REDIRECTED
S1	First Attempt	Invalid	S2
S2	Second Attempt	Invalid	S3
S3	Third Attempt	Valid	S4
S4	Home Page		
S5	Error Page		

By using the above state transition table we can perform testing of any software application. We can make a state transition table by determining desired output, and then exercise the software system to examine whether it is giving desired output or not.

All-pairs Testing

All-pairs testing technique is also known as pairwise testing. It is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input (radio button is used when you

have to select only one option for example when you select gender male or female, you can select only one option), list box, text box, etc.

Suppose, you have a function of a software application for testing, in which there are 10 fields to input the data, so the total number of discrete combinations is 10 ^ 10 (100 billion), but testing of all combinations is complicated because it will take a lot of time.

So, let's understand the testing process with an example:

Assume that there is a function with a list box that contains 10 elements, text box that can accept 1 to 100 characters, radio button, checkbox and OK button.

The input values are given below that can be accepted by the fields of the given function.

1. Check Box - Checked or

Unchecked2. List Box - 0, 1, 2, 3, 4, 5,

6, 7, 8, 9,

3. Radio Button - On or Off
4. Text Box - Number of alphabets between 1 to 100.
5. OK - Does not accept any value, only redirects to the next page.

Calculation of all the possible combinations:

1. Check Box = 2
2. List Box = 10
3. Radio Button = 2
4. Text Box = 100
5. Total number of test cases = 2*10*2*100
6. = 4000

The total number of test cases, including negative test cases, is 4000.

Testing of 4000 positive and negative test cases, is a very long and time-consuming process. Therefore, the task of the testing team is to reduce the number of test cases, to do this, the testing team considers the list box values in such a way that the first value is 0, and the other value can be any numeric number, neither positive nor negative. Ten values are now converted into 2 values.

Values of checkbox and radio button cannot be reduced because each has a combination of only 2 values. At last, the value of the text box is divided into three input categories valid integer, invalid integer, and alpha-special character.

Now, we have only 24 test cases, including negative test cases.

1. 2*2*2*3 = 24

Now, the task is to make combinations for all pair technique, into which each column should have an equal number of values, and the total value should be equal to 24.

In order to make text box column, put the most common input on the first place that is **a valid integer**, on the second place put the second most common input that is an **invalid integer**, and at the last place put the least common input that is an **Alpha Special Character**.

Then start filling the table, the first column is a text box with three values, the next column is a list box that has 2 values, the third column is a checkbox that has 2 values, and the last one is a radio button that also has 2 values.

Text box	List Box	Check Box	Radio Button
Valid Integer	0	Check	ON
Invalid Integer	Other	Uncheck	OFF
Valid Integer	0	Check	ON
Invalid Integer	Other	Uncheck	OFF
AlphaSpecialCharacter	0	Check	ON
AlphaSpecialCharacter	Other	Uncheck	OFF

In the table, we can see that the conventional software method results in 24 test cases instead of 4000 cases, and the pairwise testing method only in just 6 pair test cases.

Use Case Technique

The use case is functional testing of the black box testing used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software basedon the functionality of each function from start to end.

It is a graphic demonstration of business needs, which describe how the end-user will cooperate with the software or the application. The use cases provide us all the possible techniques of how the end-user uses the application as we can see in the below image, that how the **use case** will look like:

In the above image, we can see that a sample of a use case where we have a requirement related to the customer requirement specification (CRS).

For **module P** of the software, we have six different features.

And here, **Admin** has access to all the **six features**, the **Paid user** has access to the **three features** and for the **Free user**, there is **no access** provided to any of the features.

Like for **Admin**, the different conditions would be as below:

Pre-condition→ Admin must be generated

Action→ Login as Paid user

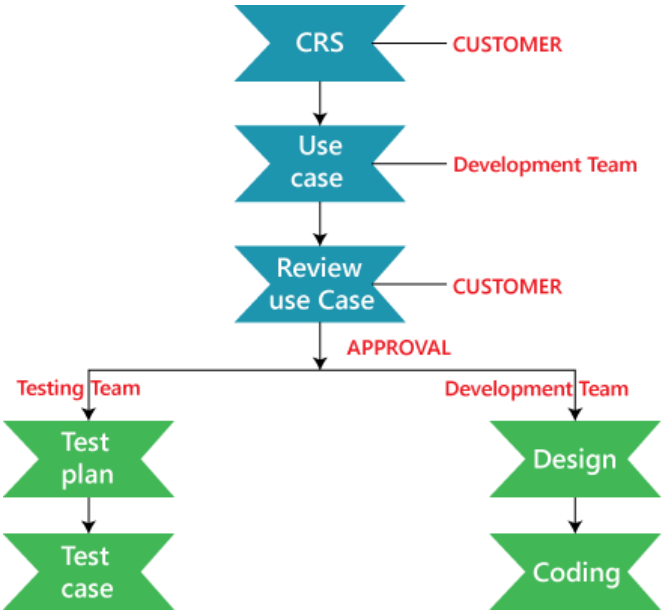
Post-condition→ 3 features must be present

And for **Free user**, the different condition would be as below:

Pre-condition→ free user must be generated

Action→ Login as a free user **Post-condition**→ no features Who writes the use case?

The client provides the customer requirement specification for the application, then the development team will write the **use case** according to the CRS, and the use case is sent to the customer for their review.



If the client approves it, then the approved **use case** is sent to the development team for further design and coding process and these approved use case is also sent to the testing team, so they can start writing the test plan and later on start writing the test cases for thedifferent features of the software.

In the below scenario, there is a tester who represents the user to use the functions of a system one by one. In this scenario, there is an actor who represents the user to use the functions of a software system.

This describes step-by-step functionality of the software application which can be understood with an example, assume that there is a software application of online moneytransfer. The various steps for transferring money are as follows:

- The user does login for the authentication of the actual user.
- The system checks ID and password with the database to ensure that whether it is a valid user or not.
- If the verification succeeds, the server connects the user to the account page, otherwise returns to the login page.
- In the account page, there are several options because the examiner is checking the money transfer option; the user goes into the money transfer option.
- After successful completion of this step, the user enters the account number in which he wants to transfer money. The user also need to enter other details like bank name, amount, IFSC code, home branch, etc.

In the last step, if there is a security feature that includes verification of the ATM cardnumber and PIN, then enter the ATM card number, PIN and other required details.

If the system is successfully following all the steps, then there is no need to design test cases for this function. By describing the steps to use, it is easy to design test cases for software systems.

How developers develop the use cases

The developers use the standard symbols to write a use case so that everyone will understand easily. They will use the **Unified modeling language** (UML) to create the usecases.

There are various tools available that help to write a use case, such as **Rational Rose.** This tool has a predefined UML symbols, we need to drag and drop them to write a use case, and the developer can also use these symbols to develop the use case.

Advantage of Use Case Technique

The use case technique gives us some features which help us to create an application.

Following are the benefits of using the use case technique while we are developing the product:

- The use case is used to take the functional needs of the system.
- These are the classification of steps, which describe the connections between the user and its actions.
- It starts from an elementary view where the system is created first and primarily used for its users.
- It is used to determine the complete analyses, which help us to achieve the complication, and then it focuses on the one detailed features at a time.

UNIT IV

TESTING TYPES

Unit Testing

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.

Smoke Testing

Smoke Testing comes into the picture at the time of receiving build software from the development team. The purpose of smoke testing is to determine whether the build software is testable or not. It is done at the time of "building software." This process is also known as "Day 0".

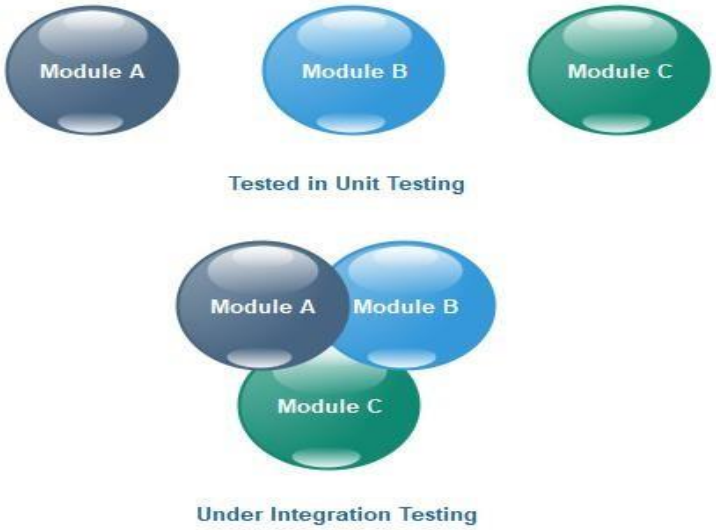
It is a time-saving process. It reduces testing time because testing is done only when the key features of the application are not working or if the key bugs are not fixed. The focus of Smoke Testing is on the workflow of the core and primary functions of the application.

Testing the basic & critical feature of an application before doing one round of deep, rigorous testing (before checking all possible positive and negative values) is known as smoke testing.

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**.

System Testing

System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software (any software is only a single element of a computer system). The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

To check the end-to-end flow of an application or the software as a user is known as **System testing**. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.

It is **end-to-end testing** where the testing environment is similar to the production environment.

Performance Testing

In this section, we will learn about performance testing, why we need it, types of performance testing, and the performance testing process.

Following are the topics, which we will understand in this section:

It is the most important part of non-functional testing.

Generally, this testing defines how quickly the server responds to the user's request.

While doing performance testing on the application, we will concentrate on the various factors like **Response time, Load, and Stability** of the application.

Response time: Response time is the time taken by the server to respond to the client's request.

Load: Here, Load means that when **N-number** of users using the application simultaneously or sending the request to the server at a time.

Stability: For the stability factor, we can say that, when N-number of users using the application simultaneously for a particular time.

We will do performance testing once the software is stable and moved to the production, and it may be accessed by the multiple users concurrently, due to this reason, some performance issues may occur. To avoid these performance issues, the tester performs one round of performance testing.

Since it is non-functional testing which doesn't mean that we always use performance testing, we only go for performance testing when the application is functionally stable.

Types of Performance Testing

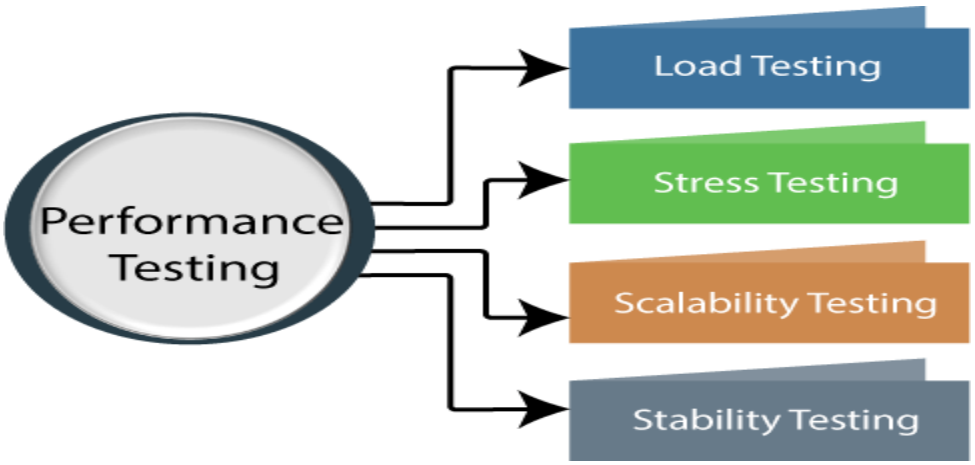
Following are the types of performance testing:

Load testing

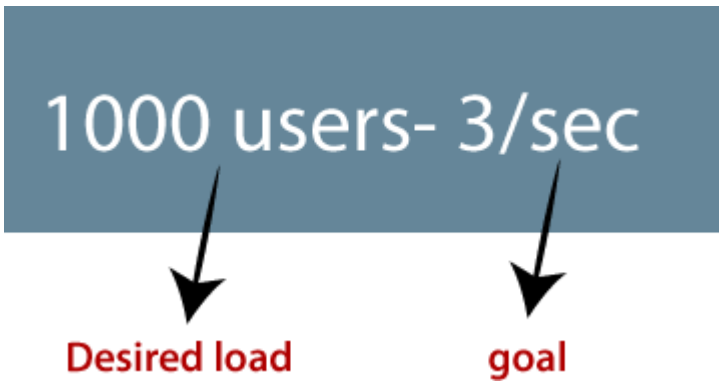
Stress testing

Scalability testing

Stability testing



Let us discuss one by one to give you a complete understanding of **Load, Stress, Scalability,** and **Stability** performance testing.



Load testing

The load testing is used to check the performance of an application by applying some load which is either less than or equal to the desired load is known as load testing.

For example: In the below image, 1000 users are the desired load, which is given by the customer, and 3/second is the goal which we want to achieve while performing a load testing.

Stress Testing

The stress testing is testing, which checks the behavior of an application by applying load greater than the desired load.

For example: If we took the above example and increased the desired load 1000 to 1100 users, and the goal is 4/second. While performing the stress testing in this scenario, it will pass because the load is greater (100 up) than the actual desired load.

Scalability Testing

Checking the performance of an application by increasing or decreasing the load in particular scales (no of a user) is known as **scalability testing**. Upward scalability and downward scalability testing are called scalability testing.

Scalability testing is divided into two parts which are as follows:

- **Upward scalability testing**
- **Downward scalability testing**

Upward scalability testing

It is testing where we **increase the number of users on a particular scale** until we get a crash point. We will use upward scalability testing to find the maximum capacity of an application.

Downward scalability testing

The downward scalability testing is used when the load testing is not passed, then start **decreasing the no. of users in a particular interval** until the goal is achieved. So that it is easy to identify the bottleneck (bug).

Stability Testing

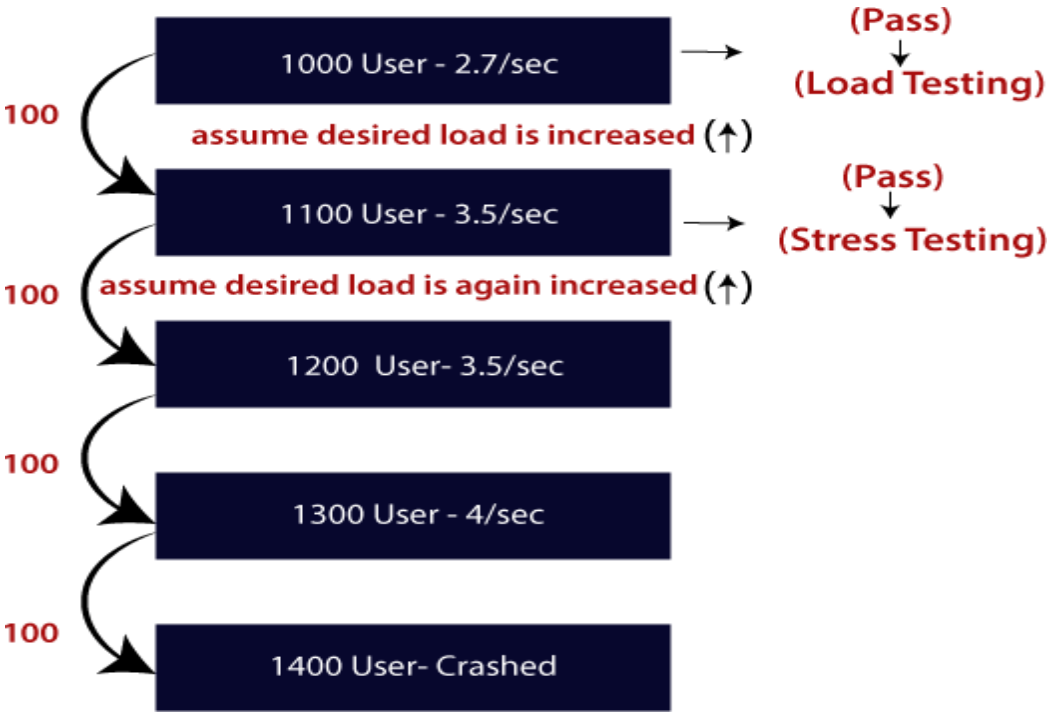
Checking the performance of an application by **applying the load for a particular duration of time** is known as **Stability Testing**.

Performance testing example

Let us take one example where we will **test the behavior of an application where the desired load is either less than 1000 or equal to 1000 users**.

In the below image, we can see that the **100 up** users are increased continuously to check the **maximum load**, which is also called **upward scalability testing**.

- **Scenario1:** When we have the 1000 users as desired load, and the 2.7/sec is goal time, these scenarios will pass while performing the load test because in load testing, we will concentrate on the no. of users, and as per the requirement it is equal to 1000 user.
- **Scenario2:** In the next scenario, we will increase the desired load by 100 users, and goal time will go up to 3.5\sec. This scenario will pass if we perform stress testing because here, the actual load is greater than (1100) the desired load (1000).
- **Scenario3:** In this, if we increase the desired load three times as **1200 → 3.5\sec:** [it is not less than or equal to the desired load that's why it will **Fail**] **1300 → 4\sec:** [it is not less than or equal to the desired load. i.e., **Fail**] **1400 → Crashed**



Security Testing

Security testing is an integral part of software testing, which is used to discover the weaknesses, risks, or threats in the software application and also help us to stop the nasty attack from the outsiders and make sure the security of our software applications.

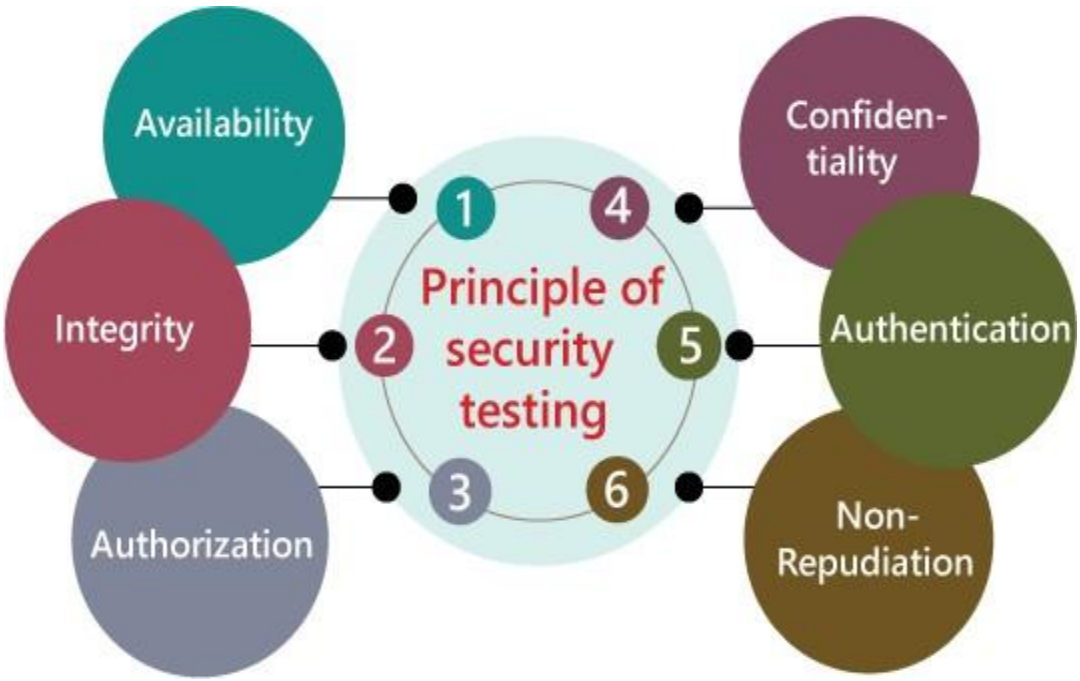
The primary objective of security testing is to find all the potential ambiguities and vulnerabilities of the application so that the software does not stop working. If we perform security testing, then it helps us to identify all the possible security threats and also help the programmer to fix those errors.

It is a testing procedure, which is used to define that the data will be safe and also continue the working process of the software.

Principle of Security testing

Here, we will discuss the following aspects of security testing:

- Availability
- Integrity
- Authorization
- Confidentiality
- Authentication
- Non-repudiation



Availability

In this, the data must be retained by an official person, and they also guarantee that the data and statement services will be ready to use whenever we need it.

Integrity

In this, we will secure those data which have been changed by the unofficial person. The primary objective of integrity is to permit the receiver to control the data that is given by the system.

The integrity systems regularly use some of the similar fundamental approaches as confidentiality structures. Still, they generally include the data for the communication to create the source of an algorithmic check rather than encrypting all of the communication. And also verify that correct data is conveyed from one application to another.

Authorization

It is the process of defining that a client is permitted to perform an action and also receive the services. The example of authorization is Access control.

Confidentiality

It is a security process that protracts the leak of the data from the outsider's because it is the only way where we can make sure the security of our data.

Authentication

The authentication process comprises confirming the individuality of a person, tracing the source of a product that is necessary to allow access to the private information or the system.

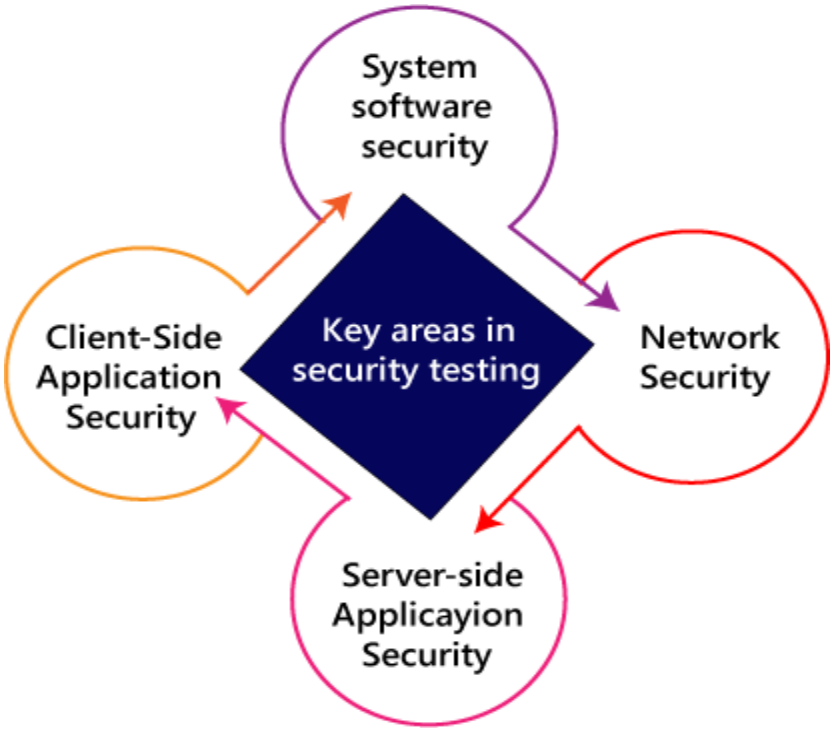
Non- repudiation

It is used as a reference to the digital security, and it a way of assurance that the sender of a message cannot disagree with having sent the message and that the recipient cannot repudiate having received the message.

The non-repudiation is used to ensure that a conveyed message has been sent and received by the person who claims to have sent and received the message.

Key Areas in Security Testing

While performing the security testing on the web application, we need to concentrate on the following areas to test the application:



System software security

In this, we will evaluate the vulnerabilities of the application based on different software such as **Operating system, Database system**, etc.

Network security

In this, we will check the weakness of the network structure, such as **policies and resources**.

Server-side application security

We will do the server-side application security to ensure that the server encryption and its tools are sufficient to protect the software from any disturbance.

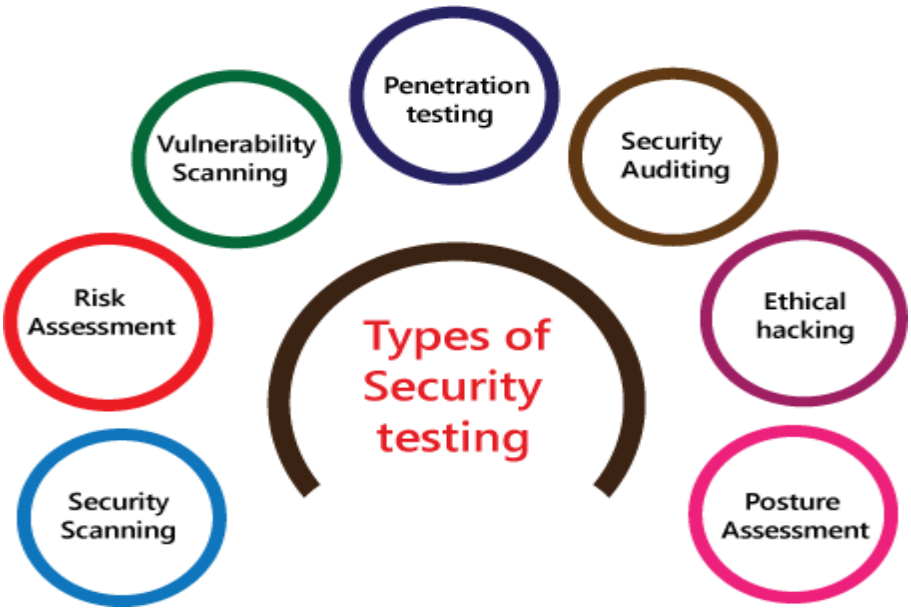
Client-side application security

In this, we will make sure that any intruders cannot operate on any browser or any tool which is used by customers.

Types of Security testing

As per Open Source Security Testing techniques, we have different types of security testing which as follows:

- **Security Scanning**
- **Risk Assessment**
- **Vulnerability Scanning**
- **Penetration testing**
- **Security Auditing**
- **Ethical hacking**
- **Posture Assessment**



Usability Testing

It is a wide testing where we need to have an application knowledge.

When we use usability testing, it makes sure that the developed software is easy while using the system without facing any problem and makes end-user life easier.

Usability testing is testing, which checks the defect in the end-user interaction of software or the product.

It is also known as User Experience (UX) Testing.

Usability testing can be done at the design stage of the Software development life cycle (SDLC), and that helps us to get more clarity of the user's needs.

Here, the user-friendliness can be described in many aspects, such as:

- Easy to understand
- Easy to access
- Look & feel
- Faster to Access
- Effective Navigation
- Good Error Handling

Easy to understand

- All the features of software or applications must be visible to the end-users.

Easy to Access

- A user-friendly application should be accessible by everyone.

Easy to Access

- The look and feel of the application should be excellent and attractive to get the user's interest.
- The GUI of the software should be good because if the GUI is not well, the user may be lost his/her interest while using the application or the software.
- The quality of the product is up to the mark as given by the client.

Faster to Access

- The software should be faster while accessing, which means that the response time of the application is quick.
- If the response time is slow, it might happen that the user got irritated. We have to ensure that our application will be loaded within 3 to 6 seconds of the response time.

Effective Navigation

Effective navigation is the most important aspect of the software.

Some of the following aspects for effective navigation:

- Good Internal Linking
- Informative header and footer
- Good search feature

Good Error Handling

- Handling of error at a coding level, make sure that the software or the application is bug-free and robust.
- By showing the correct error message will help to enhance the user experience and usability of the application.

Examples of usability testing

Let us see some examples, where we understand the use of usability testing.

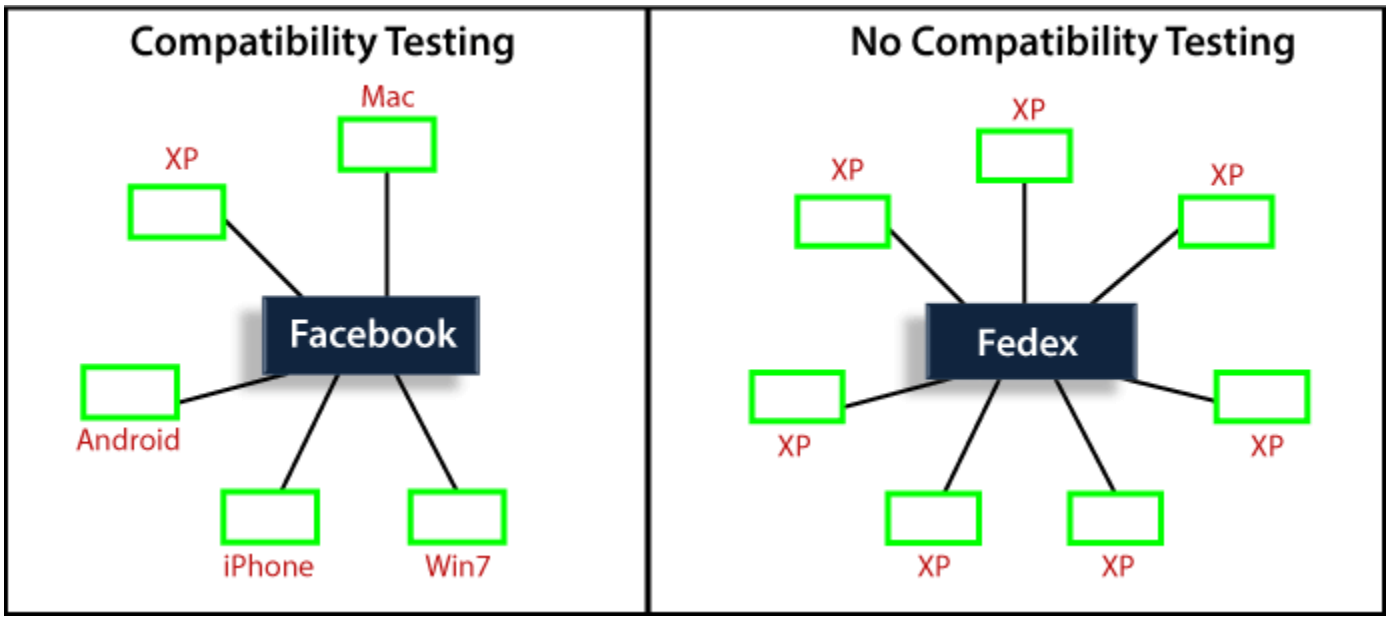
Example1:

We are taking one banking application where we produce the application for the manager.

Compatibility testing

Checking the functionality of an application on different software, hardware platforms, network, and browsers is known as compatibility testing.

Generally, we go for compatibility testing, only when the application or software is functionally stable.



Types of Compatibility testing

Following are the types of compatibility testing:

- **Software**
- **Hardware**
- **Network**
- **Mobile**

Software

Here, software means different operating systems (Linux, Window, and Mac) and also check the software compatibility on the various versions of the operating systems like Win98, Window 7, Window 10, Vista, Window XP, Window 8, UNIX, Ubuntu, and Mac.

And, we have two types of version compatibility testing, which are as follows:

- **Forward Compatibility Testing:** Test the software or application on the new or latest versions.
  For example: Latest Version of the platforms (software)
  Win 7 → Win 8 → Win 8.1 → Win 10
- **Backward Compatibility Testing:** Test the software or application on the old or previous versions.
  For example: Window XP → Vista → Win 7 → Win 8 → Win 8.1

And different browsers like **Google Chrome, Firefox,** and **Internet Explorer**, etc.

Hardware

The application is compatible with different sizes such as RAM, hard disk, processor, and the graphic card, etc.

Mobile

Check that the application is compatible with mobile platforms such as iOS, Android, etc.

Network

Checking the compatibility of the software in the different network parameters such as operating speed, bandwidth, and capacity.

AdHoc Testing

When a software testing performed without proper planning and documentation, it is said to be Adhoc Testing. Such kind of tests are executed only once unless we uncover the defects.

Adhoc Tests are done after formal testing is performed on the application. Adhoc methods are the least formal type of testing as it is NOT a structured approach. Hence, defects found using this method are hard to replicate as there are no test cases aligned for those scenarios.

Testing is carried out with the knowledge of the tester about the application and the tester tests randomly without following the specifications/requirements. Hence the success of Adhoc testing depends upon the capability of the tester, who carries out the test. The tester has to find defects without any proper planning and documentation, solely based on tester's intuition.

Forms of Adhoc Testing :

1. Buddy Testing: Two buddies, one from development team and one from test team mutually work on identifying defects in the same module. Buddy testing helps the testers develop better test cases while development team can also make design changes early. This kind of testing happens usually after completing the unit testing.
2. Pair Testing: Two testers are assigned the same modules and they share ideas and work on the same systems to find defects. One tester executes the tests while another tester records the notes on their findings.
3. Monkey Testing: Testing is performed randomly without any test cases in order to break the system.

Various ways to make Adhoc Testing More Effective

1. Preparation: By getting the defect details of a similar application, the probability of finding defects in the application is more.
2. Creating a Rough Idea: By creating a rough idea in place the tester will have a focussed approach. It is NOT required to document a detailed plan as what to test and how to test.
3. Divide and Rule: By testing the application part by part, we will have a better focus and better understanding of the problems if any.
4. Targeting Critical Functionalities: A tester should target those areas that are NOT covered while designing test cases.
5. Using Tools: Defects can also be brought to the lime light by using profilers, debuggers and even task monitors. Hence being proficient in using these tools one can uncover several defects.
6. Documenting the findings: Though testing is performed randomly, it is better to document the tests if time permits and note down the deviations if any. If defects are found, corresponding test cases are created so that it helps the testers to retest the scenario.

Internationalization testing

Internationalization testing is the process of verifying the application under test to work uniformly across multiple regions and cultures.

The main purpose of internationalization is to check if the code can handle all international support without breaking functionality that might cause data loss or data integrity issues. Globalization testing verifies if there is proper functionality of the product with any of the locale settings.

Internationalization Checklists:

1. Testing to check if the product works across settings.
2. Verifying the installation using various settings.
3. Verify if the product works across language settings and currency settings.

SOA

SOA is a method of integrating business applications and processes together so as to meet the business needs.

In Software Engineering, SOA provides agility and flexibility to business processes. The changes to the process or application can be directed to a particular component without affecting the whole system.

The software developers in SOA either develop or buy chunks of programs called **SERVICES.**

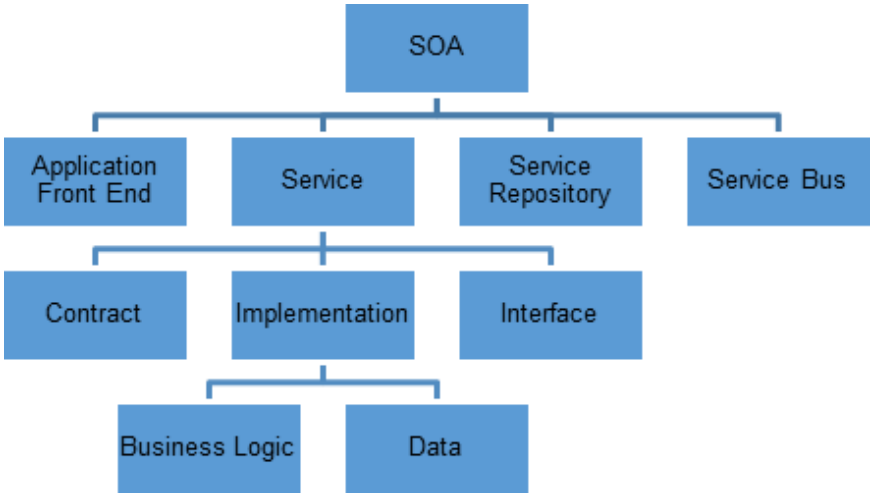
- Services can be a functional unit of application or business process, which can be reused or repeated by any other application or process.

  (For example, in the above image, Payment Gateway is a service which can be reused by any e-commerce site. Whenever a payment needs to do, the e-commerce site calls/Requests the Payment Gateway service. After payment is done on a gateway, a response is sent to the e-commerce website)

- Services are easy to assemble and easy to reconfigure components.
- Services can be compared to building blocks. They can construct any application needed. Adding and removing them from the application or business process is easy.
- Services are defined more by the business function they perform rather than as chunks of code.

SOA Testing

SOA consists of various technologies. Applications built using SOA has various services which are loosely coupled.



SOA Testing should focus on 3 system layers

Services Layer

This layer consists of the services, services exposed by a system derived from business functions.

For example –

Consider a Wellness Website which consists of

Weight Tracker

Blood Sugar Tracker

Blood Pressure Tracker

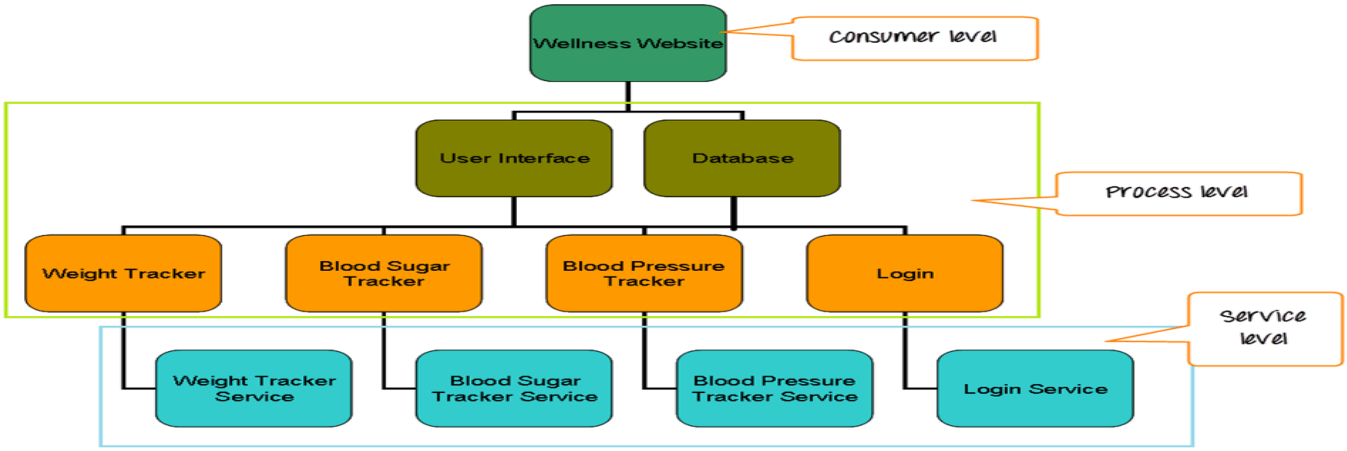
- Trackers display the respective data and date they are entered. Services layer consists of the services which gets the respective data from the Database–
- Weight Tracker service
- Blood Sugar Tracker service
- Blood Pressure Tracker service
- Login Service

Process Layer

- Process Layer consists the processes, collection of services which are part of a single functionality.
- The processes might be a part of user interface (for ex – A search engine), a part of an ETL tool (for getting data from the database).
- The main focus in this layer will be in user interfaces and process.
- The user interface of the weight tracker and its integration with the Database is the primary focus.
- Below functions will be of consideration
- Adding new data
- Editing existing data
- Creating new tracker
- Deleting data

Consumer Layer

This layer mainly comprises of user interfaces



Based on the layer, the testing of an SOA application is distributed into three levels.

- Service level
- Interface level
- End to End level

Top Down approach is used for Test Designing.

Bottom Up approach is used for Test Execution.

Functional Testing of Mobile Application

- The **Functional Testing of Mobile Application** is a process of testing functionalities of mobile applications like user interactions as well as testing the transactions that users might perform. The main purpose of mobile application functional testing is to ensure the quality, meeting the specified expectations, reducing the risk or errors and customer satisfaction.
- The various factors which are relevant in functional testing are
- Type of application based upon the business functionality usages (banking, gaming, social or business)
- Target audience type (consumer, enterprise, education)
- Distribution channel which is used to spread the application (e.g. Apple App Store, Google play, direct distribution)

The most fundamental test scenarios in the functional testing can be considered as :

- To validate whether all the required mandatory fields are working as required.

- To validate that the mandatory fields are displayed in the screen in a distinctive way than the non-mandatory fields.
- To validate whether the application works as per as requirement whenever the application starts/stops.
- To validate whether the application goes into minimized mode whenever there is an incoming phone call. In order to validate the same we need to use a second phone, to call the device.
- To validate whether the phone is able to store, process and receive SMS whenever the app is running. In order to validate the same we need to use a second phone to send sms to the device which is being tested and where the application under test is currently running.
- To validate that the device is able to perform required multitasking requirements whenever it is necessary to do so.
- To validate that the application allows necessary social network options such as sharing, posting and navigation etc.
- To validate that the application supports any payment gateway transaction such as Visa, Mastercard, Paypal etc as required by the application.
- To validate that the page scrolling scenarios are being enabled in the application as necessary.
- To validate that the navigation between relevant modules in the application are as per the requirement.
- To validate that the truncation errors are absolutely to an affordable limit.
- To validate that the user receives an appropriate error message like “Network error. Please try after some time” whenever there is any network error.
- To validate that the installed application enables other applications to perform satisfactorily, and it does not eat into the memory of the other applications.
- To validate that the application resumes at the last operation in case of a hard reboot or system crash.
- To validate whether the installation of the application can be done smoothly provided the user has the necessary resources and it does not lead to any significant errors.
- To validate that the application performs auto start facility according to the requirements.
- To validate whether the application performs according to the requirement in all versions of Mobile that is 2g, 3g and 4g.
- To perform Regression Testing to uncover new software bugs in existing areas of a system after changes have been made to them. Also rerun previously performed tests to determine that the program behavior has not changed due to the changes.

- To validate whether the application provides an available user guide for those who are not familiar to the app

UNIT V

TEST CASE DESIGN

Test Case Definition

A Test case is a group of steps that is to be executed to check the functionality of a specific object or an operational logic. Test case describes the user input and the system response with some preconditions to determine the correctness and completeness of the functionality.

Test case is usually associated with at least one business function/requirement that is being validated. It requires a specific test data to be developed for input during , test case execution. The test case execution may be governed by preconditions than are required or setup before the execution of the test case, such as database support, printer setup, or data that should exist at the start of the test case execution. A test case is a self-standing and self-cleaning test with a specific purpose that points back to the specifications, and if failed, it points to the bug.

Finally, we conclude that a test case validates one or more criteria to certify that an application is structurally and functionally ready to be implemented into the business production environment

Necessity of Test Case Documentation

Test case design determines the strategies of operating, observing, and evaluating the software build. Sometimes, a proper test design can even detect mix-ups in already furnished designs of the application, functional specifications, or the use cases. Hence, designing test cases is essential to perform a good test. A good test is the one that has high probability of finding an error.

The importance of test case documentation is as follows:

- To assure that the entire functionality of the product is covered
- To maintain the status of the testing process, test results, etc.
- To build the traceability matrix (using which we can map use case to the bug id via test case id)

- To prepare effort estimation
- For reference to work on the upgraded versions of the product in future
- To compare with or use a similar product

Rules to be Followed rules.

The test cases will be effective and reusable, it they are designed by following certain rules. Therefore, a test case must

- Be designed such that they are reusable
- Give consistent results independent of testers
- Specify what the tester has to perform and the response of the system
- Be specific to the testing element or logic, etc.
- Be divided into different test cases to keep them short and avoid confusions within the testers
- Be divided into positive and negative scenarios
- Be uniquely identified for revisiting
- Hold the bug id for tacking purpose
- Start with the word 'Verify' or 'Check'
- Contain a maximum of 10 to 15 steps
- Be reviewed by the superiors to ensure the coverage of whole functionality
- Be approved by the client

Test case Design Methods

Test cases can be designed based on the following:

- Functional specifications based test cases design
- Use cases based test cases design
- Application based test cases design

Functional Specification Based Test Case Design

Usually the project management supplies this functional specification document. This type of testing activity checks whether the product is working as per the functional specifications

or not. We should be able to decide the different types of tests that are to be per formed on the product, such as functional testing, performance testing, and other system tests.

In this method, the client interaction would be critical to clarify the issues on the functionality, relations, and others. It is rare to have a complete set o f functional specifications. Hence, to author test cases using functional specifications, one should follow a procedure:

- Analyze the requirements in terms of entry point, exit point, normal flow, alternative flows, input required, expected outputs, and exceptions
- Analyze the dependency in between requirements.
- Understand the behavior by exploring the corresponding application/product
- Determine the order or sequence of requirements in which specific transact must be tested
- Determine the preconditions or setup necessary to execute a test procedure, as put the application build in ready state, database configuration, or requirements
- Determine the high-risk requirements
- Author the test case titles or test scenarios
- Review the test case titles or test scenarios
- Finally, write the test cases documents

Functional Specification Example#1

A login process allows user id and password to authorize users. To authorize, users must give alphanumeric in lowercase from 4 to 16 characters long for user id and alphabets in lowers case from 4 to 8 characters long for password

Test cases: Verify user-id entry

BVA		
Min	4 chars	Pass
Min-1	3 chars	fail
Min+1	5 chars	Pass
Max	16 chars	Pass
Max+1	17 chars	Fail
Max-1	15chars	pass

ECP	
Valid	In valid
	A to Z Special characters, Blank
0 to 9	

Test case 2:Verify password Entry

BVA		
Min	4 chars	Pass
Max	8 chars	pass
Max+1	9 chars	fail
Min-1	3 chars	Fail

ECP	
Valid	In valid
a to z	A to Z Special characters, Blank ,0 to 9

Test case 3:Login operation (test procedure)

User ID	Password	Criteria
Valid	Valid	Pass
Valid	Invalid	Fail
Invalid	Valid	Fail
Value	Blank	Fail

Blank	value	Fail

Use cases based test cases design

A use case is a sequence of transactions that gives a measurable result of value for an actor. Actor is a person, thing, or any external application that uses the application under development or on which the current application may depend. The collection of use cases is the system's complete functionality. A use case defines a goal-oriented set of Interactions between external users and the system under consideration or development.

A use case scenario is a description that illustrates, a systematic process of how a user intends to use a system, essentially capturing the system behavior from the user's point of view. Usually, during Requirements gathering phase, use cases are captured from the client

Use Case Name	Name of use case for future reference
Use Case description	Summary of functionality, which is described in this use case
Actors	Name of the actors, which are participating in this use case covering functionalities
Related Use Cases	Names of related use cases, which have dependency with this use case covering functionality
Preconditions	List of necessary actions to do before operating this use case covering functionality
Activity flow diagram	The diagram provides an understandable flow of functionality with inputs, tasks and outputs
Primary scenario	A step-by-step list of actions from base action to end action of that functionality
Alternative Scenario	An alternative list of actions from base action to end action of that functionality, if possible
Post conditions	It specifies exit criteria of that functionality
UI Mock up	A model screen or prototype

Writing Good Test Cases

- As far as possible, write test cases in such a way that you test only one thing at a time. Do not overlap or complicate test cases. Attempt to make your test cases 'atomic'.
- Ensure that all positive scenarios AND negative scenarios are covered.
- Language:
- Write in simple and easy-to-understand language.
- Use active voice instead of passive voice: Do this, do that.
- Use exact and consistent names (of forms, fields, etc).
- Characteristics of a good test case:
- *Accurate*: Exacts the purpose.
- *Economical*: No unnecessary steps or words.
- *Traceable*: Capable of being traced to requirements.
- *Repeatable*: Can be used to perform the test over and over.
- *Reusable*: Can be reused if necessary.

ISTQB Definition

- **test case:** A set of preconditions, inputs, actions (where applicable), expected results and postconditions, developed based on test conditions.
- **high-level test case:** A test case with abstract preconditions, input data, expected results, postconditions, and actions (where applicable).
- **low-level test case:** A test case with concrete values for preconditions, input data, expected results and postconditions and detailed description of actions (where applicable).

TEST CASE TEMPLATE

A test case can have the following elements. Note, however, that a test management tool is normally used by companies and the format is determined by the tool used.

Test Suite ID	The ID of the test suite to which this test case belongs.
Test Case ID	The ID of the test case.
Test Case Summary	The summary / objective of the test case.
Related Requirement	The ID of the requirement this test case relates/traces to.
Prerequisites	Any prerequisites or preconditions that must be fulfilled prior to executing the test.
Test Script / Procedure	Step-by-step procedure to execute the test.
Test Data	The test data, or links to the test data, that are to be used while conducting the test.
Expected Result	The expected result of the test.
Actual Result	The actual result of the test; to be filled after executing the test.
Status	Pass or Fail. Other statuses can be 'Not Executed' if testing is not performed and 'Blocked' if testing is blocked.

Remarks	Any comments on the test case or test execution.
Created By	The name of the author of the test case.
Date of Creation	The date of creation of the test case.
Executed By	The name of the person who executed the test.
Date of Execution	The date of execution of the test.
Test Environment	The environment (Hardware/Software/Network) in which the test was executed

Test case review process

When the test engineer writes a test case, he/she may skip some scenarios, inputs and writes wrong navigation steps, which may affect the entire test execution process.

To avoid this, we will do one round of **review and approval process** before starting test execution.

If we don't go for the review process, we **miss out some scenarios, accuracy won't be there, and the test engineer won't be serious**

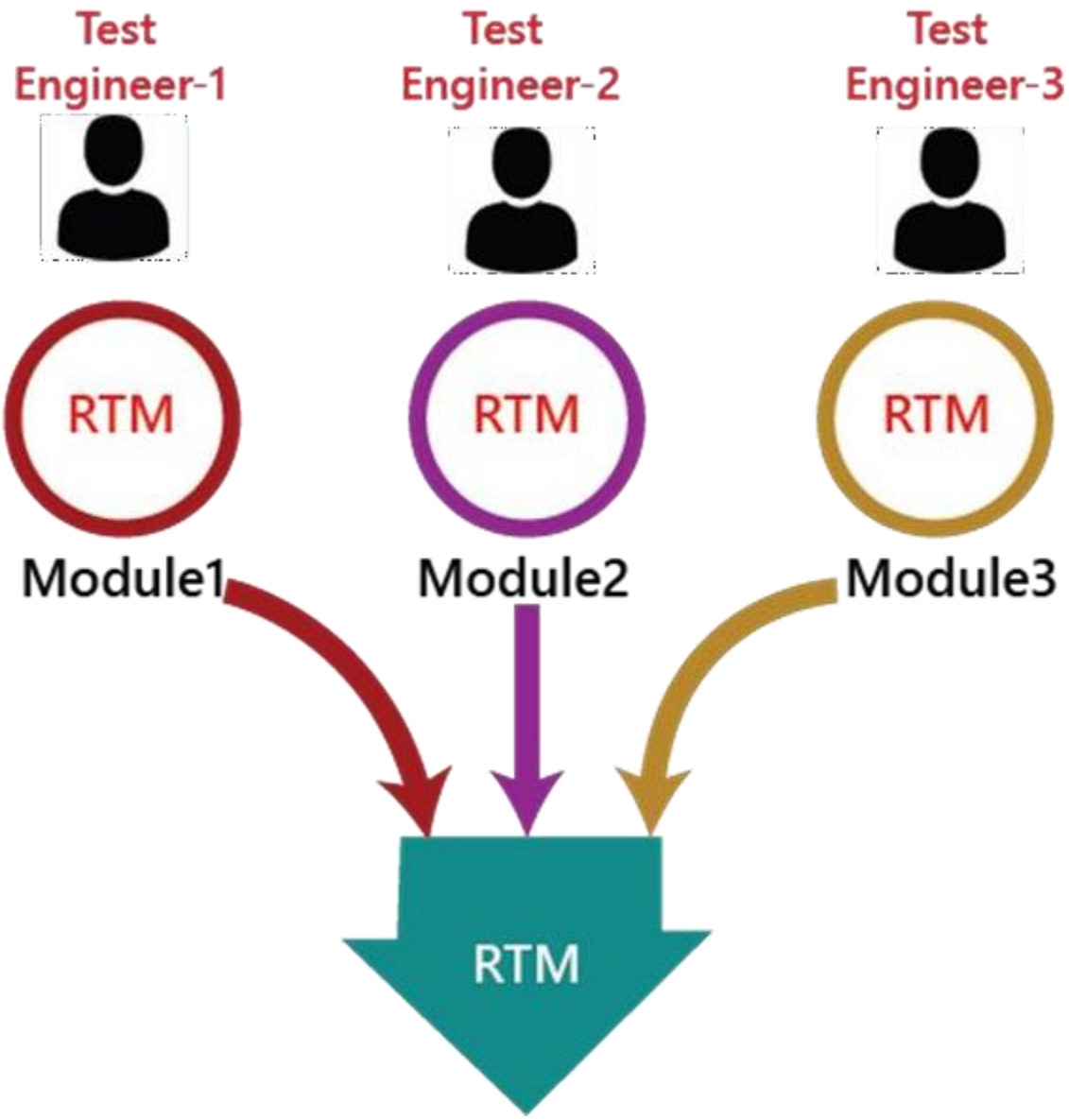
Traceability Matrix

Traceability matrix is a table type document that is used in the development of software application to trace requirements. It can be used for both forward (from Requirements to Design or Coding) and backward (from Coding to Requirements) tracing. It is also known as **Requirement Traceability Matrix (RTM) or Cross Reference Matrix (CRM).**

It is prepared before the test execution process to make sure that every requirement is covered in the form of a Test case so that we don't miss out any testing. In the RTM document, we map all the

requirements and corresponding test cases to ensure that we have written all the test cases for each condition.

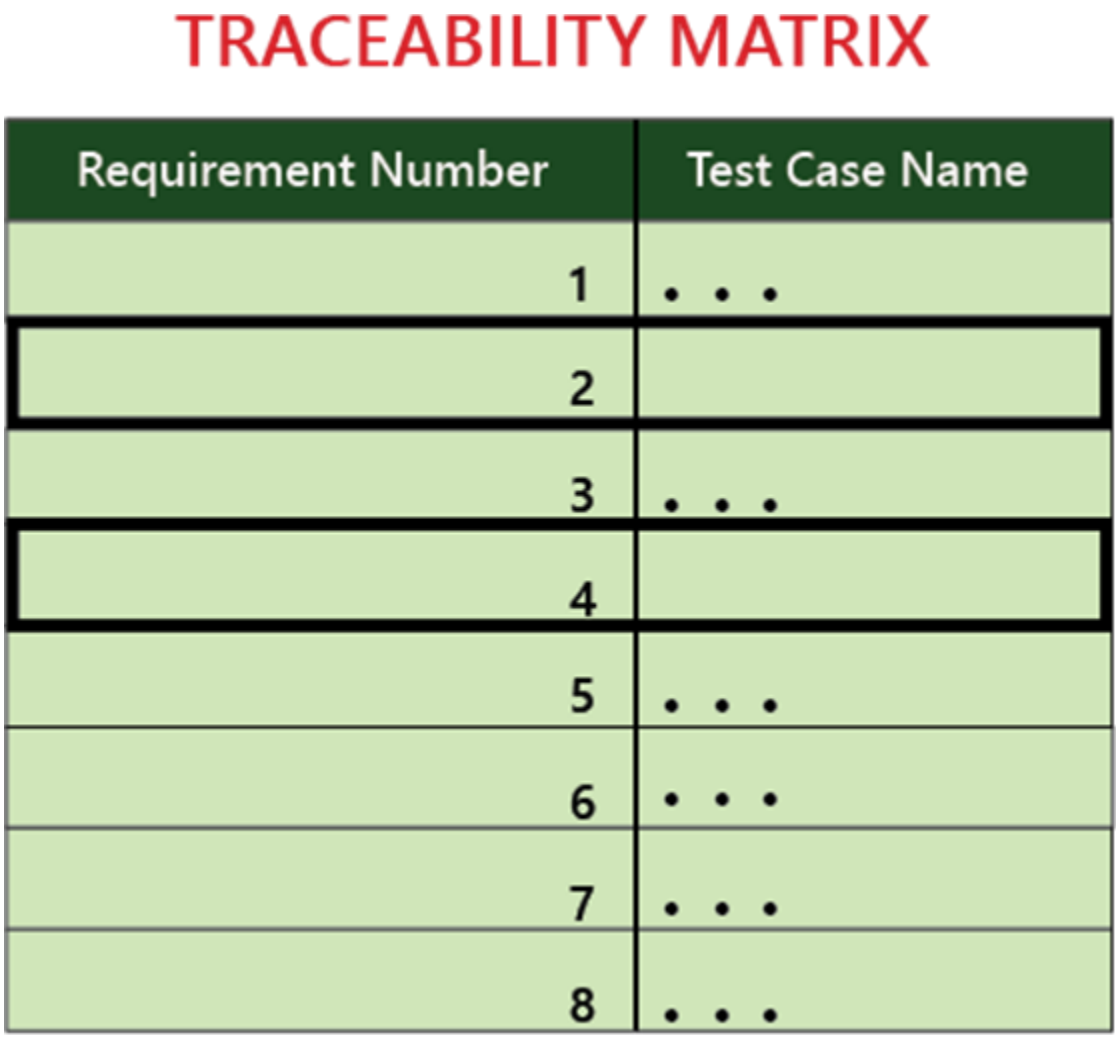
The test engineer will prepare RTM for their respective assign modules, and then it will be sent to the Test Lead. The Test Lead will go repository to check whether the Test Case is there or not and finally Test Lead consolidate and prepare one necessary RTM document.



This document is designed to make sure that each requirement has a test case, and the test case is written based on business needs, which are given by the client. It will be performed with the help of the test cases if any requirement is missing, which means that the test case is not written for a

particular need, and that specific requirement is not tested because it may have some bugs. The traceability is written to make sure that the entire requirement is covered.

We can observe in the below image that the requirement number 2 and 4 test case names are not mentioned that's why we highlighted them, so that we can easily understand that we have to write the test case for them.



Generally, this is like a worksheet document, which contains a table, but there are also many user-defined templates for the traceability matrix. Each requirement in the traceability matrix is connected with its respective test case so that tests can be carried out sequentially according to specific requirements.

Note:

We go for RTM after approval and before execution so that we don't miss out on any Test Case for any requirement.

We don't write RTM while writing the testing because it can be incomplete, and after writing the test case, we don't go here because the test case can be rejected.

RTM document ensures that at least there is one test case written in each requirement, whereas it does not talk about all possible test cases written for the particular requirement.

RTM Template

Below is the sample template of requirement traceability matrix (RTM):

Requirement no	Module name	High level requirement	Low level requirement	Test case name

Example of RTM template

Let us one sample of RTM template for better understanding:

	A	B	C	D	E
1	RTM Template				
2	**Requirement number**	**Module number**	**High level requirement**	**Low level requirement**	**Test case name**
3	2	Loan	**2.1 Personal loan**	2.1.1--> personal loan for private employee	beta-2.0-personal loan
4				2.1.2--> personal loan for government employee	
5				2.1.3--> personal loan for jobless people	
6			**2.2 Car loan**	2.2.1--> car loan for private employee	
7				——	
8			**2.3 Home loan**	——	
9				——	
10				——	
11					

Goals of Traceability Matrix

- It helps in tracing the documents that are developed during various phases of SDLC.
- It ensures that the software completely meets the customer's requirements.
- It helps in detecting the root cause of any bug.

Types of Traceability Test Matrix

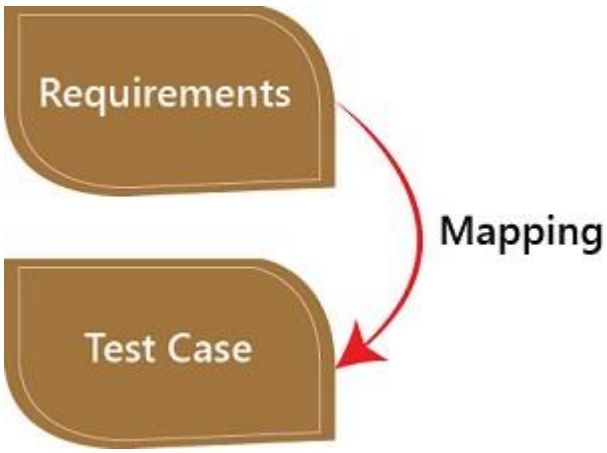
The traceability matrix can be classified into three different types which are as follows:

- Forward traceability
- Backward or reverse traceability
- Bi-directional traceability

Forward traceability

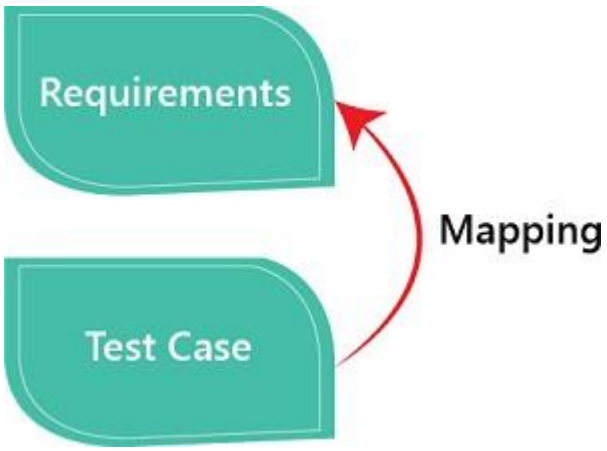
The forward traceability test matrix is used to ensure that every business's needs or requirements are executed correctly in the application and also tested rigorously. The main objective of this is to

verify whether the product developments are going in the right direction. In this, the requirements are mapped into the forward direction to the test cases.



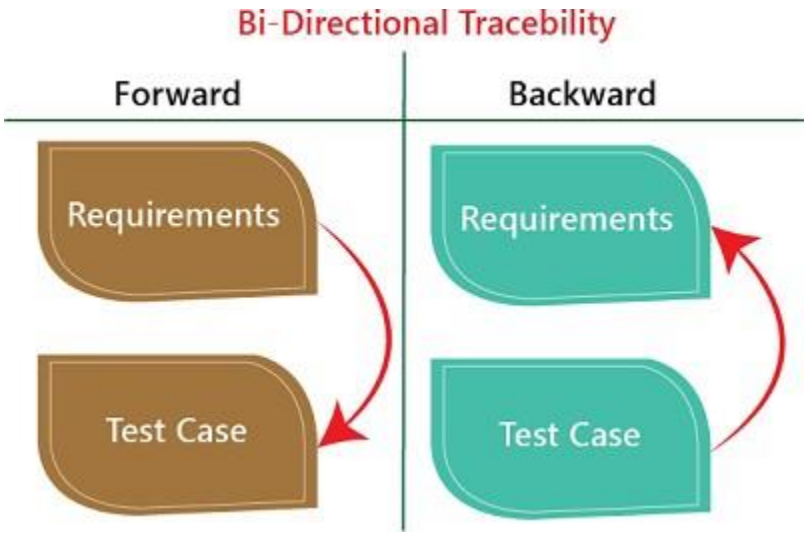
Backward or reverse traceability

The reverse or backward traceability is used to check that we are not increasing the space of the product by enhancing the design elements, code, test other things which are not mentioned in the business needs. And the main objective of this that the existing project remains in the correct direction. In this, the requirements are mapped into the backward direction to the test cases.



Bi-directional traceability

It is a combination of forwarding and backward traceability matrix, which is used to make sure that all the business needs are executed in the test cases. It also evaluates the modification in the requirement which is occurring due to the bugs in the application.



Advantage of RTM

Following are the benefits of requirement traceability matrix:

- With the help of the RTM document, we can display the complete test execution and bugs status based on requirements.
- It is used to show the missing requirements or conflicts in documents.
- In this, we can ensure the complete test coverage, which means all the modules are tested.
- It will also consider the efforts of the testing teamwork towards reworking or reconsidering on the test cases.

What is a bug in software testing?

The Bug is the informal name of defects, which means that software or application is not working as per the requirement.

In software testing, a software bug can also be issue, error, fault, or failure. The bug occurred when developers made any mistake or error while developing the product.

While testing the application or executing the test cases, the test engineer may not get the expected result as per the requirement. And the bug had various names in different companies such as error, issues, problem, fault, and mistake, etc.

Basic terminology of defect

Let see the different terminology of defect:

- **Defect**
- **Bug**
- **Error**
- **Issue**
- **Mistakev**
- **Failurev**

Terms	Description	Raised by
Defect	When the application is not working as per the requirement.	Test Engineer
Bug	Informal name of defect	Test Engineer
Error	Problem in code leads to the errors.	Developer, Automation Test Engineer
Issue	When the application is not meeting the business requirement.	Customer
Mistake	Problem in the document is known as a mistake.	--
Failure	Lots of defect leads to failure of the software.	--

Why defect/bug occur?

In software testing, the bug can occur for the following reasons:

- Wrong coding
- Missing coding
- Extra coding

Wrong coding

Wrong coding means improper implementation.

For example: Suppose if we take the Gmail application where we click on the **"Inbox"** link, and it navigates to the **"Draft"** page, this is happening because of the wrong coding which is done by the developer, that's why it is a bug.

Missing coding

Here, missing coding means that the developer may not have developed the code only for that particular feature.

For example: if we take the above example and open the inbox link, we see that it is not there only, which means the feature is not developed only.

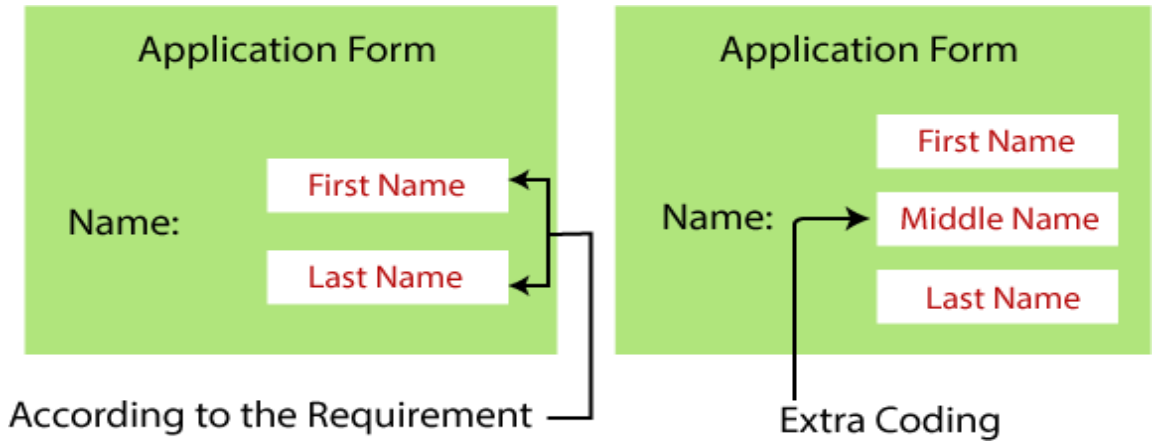
Extra coding

Here, extra coding means that the developers develop the extra features, which are not required according to the client's requirements.

For example:

Suppose we have one application form wherein the **Name field**, the **First name**, and **Last name** textbox are needed to develop according to the client's requirement.

But, the developers also develop the **"Middle name"** textbox, which is not needed according to the client's requirements as we can see in the below image:



If we develop an extra feature that is not needed in the requirement, it leads to unnecessary extra effort. And it might also happen that adding up the extra feature affects the other elements too.

Bug tracking tool

We have various types of bug tracking tools available in software testing that helps us to track the bug, which is related to the software or the application.

Some of the most commonly used bug tracking tools are as follows:

- **Jira**
- **Bugzilla**
- **Redmine**
- **Mantis**
- **Backlog**

Jira

Jira is one of the most important bug tracking tools. Jira is an open-source tool that is used for bug tracking, project management, and issue tracking in manual testing.

Jira includes different features like reporting, recording, and workflow. In Jira, we can track all kinds of bugs and issues, which are related to the software and generated by the test engineer.

To get the complete details about Jira tool, refer to the below link:

https://www.javatpoint.com/jira-tutorial

Bugzilla

Bugzilla is another important bug tracking tool, which is most widely used by many organizations to track the bugs.

Bugzilla is an open-source tool, which is used to help the customer, and the client to maintain the track of the bugs.

It is also used as a test management tool because, in this, we can easily link other test case management tools such as ALM, quality Centre, etc.

Bugzilla supports various operating systems such as Windows, Linux, and Mac.

Bugzilla has some features which help us to report the bug easily:

- A bug can be list in multiple formats
- Email notification controlled by user preferences.
- Advanced searching capabilities
- Excellent security
- Time tracking

Redmine

It is an open-source tool which is used to track the issues and web-based project management tool. Redmine tool is written in **Ruby** programing language and also compatible with multiple databases like MySQL, Microsoft SQL, and SQLite.

While using the Redmine tool, users can also manage the various project and related subprojects.

Some of the common characteristics of Redmine tools are as follows:

- Flexible role-based access control
- Time tracking functionality
- A flexible issue tracking system
- Feeds and email notification
- Multiple languages support (Albanian, Arabic, Dutch, English, Danish and so on)

MantisBT

MantisBT stands for **Mantis Bug Tracker**. It is a web-based bug tracking system, and it is also an open-source tool.

MantisBT is used to follow the software defects. It is executed in the PHP programing language.

Some of the common features of MantisBT are as follows:

- Full-text search
- Audit trails of changes made to issues
- Revision control system integration
- Revision control of text fields and notes
- Notifications
- Plug-ins
- Graphing of relationships between issues

Backlog

The backlog is widely used to manage the IT projects and track the bugs. It is mainly built for the development team for reporting the bugs with the complete details of the issues, comments. Updates and change of status. It is a project management software.

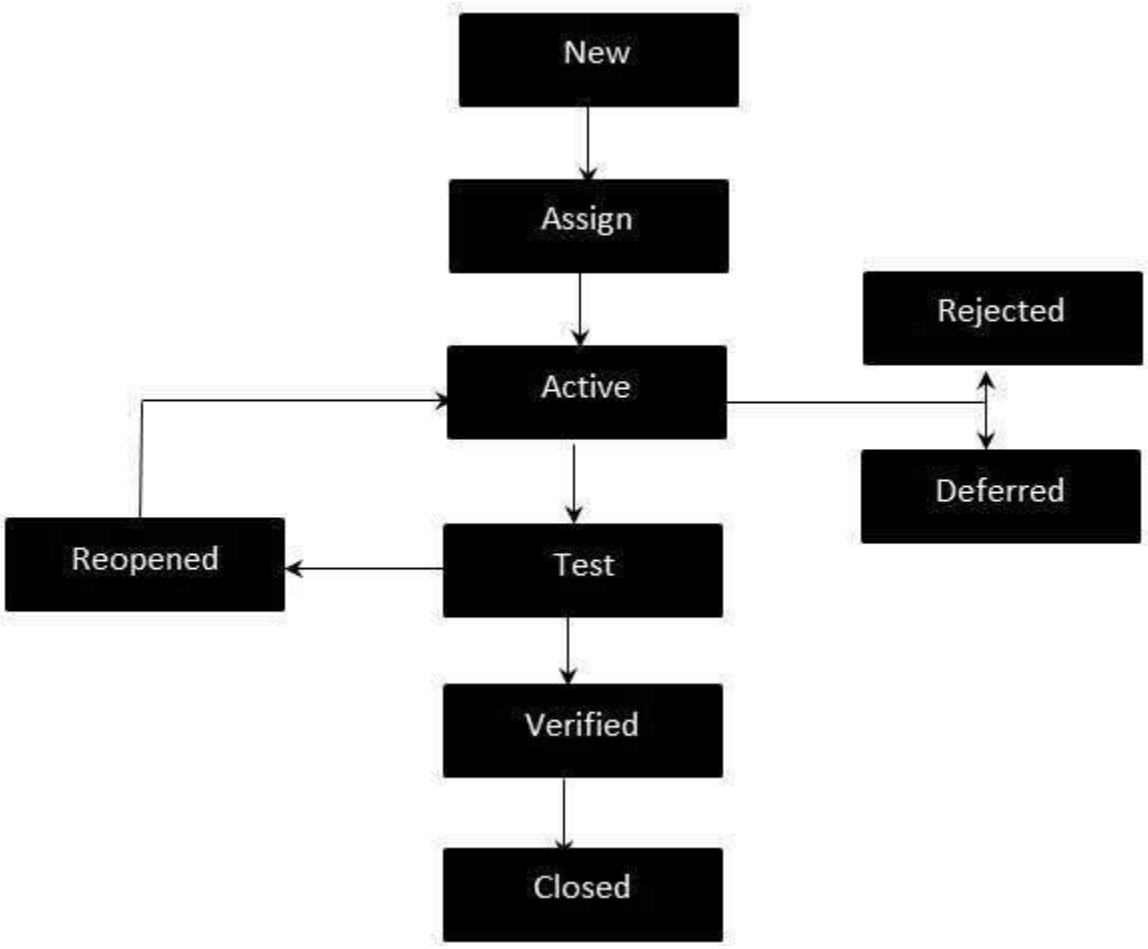
Features of backlog tool are as follows:

- Gantt and burn down charts
- It supports Git and SVN repositories
- IP access control
- Support Native iOS and Android apps

Defect Life Cycle

Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.

Defect Life Cycle - Workflow:



Defect Life Cycle States:

- New - Potential defect that is raised and yet to be validated.
- Assigned - Assigned against a development team to address it but not yet resolved.
- Active - The Defect is being addressed by the developer and investigation is under progress. At this stage there are two possible outcomes; viz - Deferred or Rejected.
- Test - The Defect is fixed and ready for testing.
- Verified - The Defect that is retested and the test has been verified by QA.
- Closed - The final state of the defect that can be closed after the QA retesting or can be closed if the defect is duplicate or considered as NOT a defect.
- Reopened - When the defect is NOT fixed, QA reopens/reactivates the defect.
- Deferred - When a defect cannot be addressed in that particular cycle it is deferred to future release.
- Rejected - A defect can be rejected for any of the 3 reasons; viz - duplicate defect, NOT a Defect, Non Reproducible.

Test execution

Test execution is the process of executing the code and comparing the expected and actual results. Following factors are to be considered for a test execution process:

- Based on a risk, select a subset of test suite to be executed for this cycle.
- Assign the test cases in each test suite to testers for execution.
- Execute tests, report bugs, and capture test status continuously.
- Resolve blocking issues as they arise.
- Report status, adjust assignments, and reconsider plans and priorities daily.
- Report test cycle findings and status.

Risk Based Testing

- **Risk Based Testing (RBT)** is a software testing type which is based on the probability of risk. It involves assessing the risk based on software complexity, criticality of business, frequency of use, possible areas with Defect etc. Risk based testing prioritizes testing of features and functions of the software application which are more impactful and likely to have defects.
- Risk is the occurrence of an uncertain event with a positive or negative effect on the measurable success criteria of a project. It could be events that have occurred in the past or current events or something that could happen in the future. These uncertain events can have an impact on the cost, business, technical and quality targets of a project.
- Risks can be positive or negative.
- **Positive risks** are referred to as opportunities and help in business sustainability. For example investing in a New project, Changing business processes, Developing new products.
- **Negative Risks** are referred to as threats and recommendations to minimize or eliminate them must be implemented for project success.
- When to implement Risk based Testing
- Risk based testing can be implemented in
- Projects having time, resource, budget constraints, etc.

- Projects where risk based analysis can be used to detect vulnerabilities to SQL injection attacks.
- Security Testing in Cloud Computing Environments.
- New projects with high risk factors like Lack of experience with the technologies used, Lack of business domain knowledge.
- Incremental and iterative models, etc.