UNIT 3

Understanding Testing

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.

User Interface design

- The visual part of a computer application or operating system through which a client interacts with a computer or software.

- It determines how commands are given to the computer or the program and how data is displayed on the screen.

Types of User Interface

- Text-Based User Interface or Command Line Interface

- Graphical User Interface (GUI)

Text-Based User Interface

This method relies primarily on the keyboard. A typical example of this is UNIX.

Advantages

- Many and easier to customizations options.
- Typically capable of more important tasks.

Disadvantages

- Relies heavily on recall rather than recognition.
- Navigation is often more difficult.

(GUI)

f this type of

Characteristics	Descriptions
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files. On other icons describes processes.
Menus	Commands are selected from a menu rather than typed in a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interests in a window.
Graphics	Graphics elements can be mixed with text or the same display.

Advantages

- Less expert knowledge is required to use it.
- Easier to Navigate and can look through folders quickly in a guess and check manner.
- The user may switch quickly from one task to another and can interact with several different applications.

Disadvantages

- Typically decreased options.
- Usually less customizable. Not easy to use one button for tons of different variations.

UI Design principles

Structure: Design should organize the user interface purposefully, in the meaningful and usual based on precise, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another.

The structure principle is concerned with overall user interface architecture.

Simplicity: The design should make the simple, common task easy, communicating clearly and directly in the user's language, and providing good shortcuts that are meaningfully related to longer procedures.

Visibility: The design should make all required options and materials for a given function visible without distracting the user with extraneous or redundant data.

Feedback: The design should keep users informed of actions or interpretation, changes of state or condition, and bugs or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

Tolerance: The design should be flexible and tolerant, decreasing the cost of errors and misuse by allowing undoing and redoing while also preventing bugs wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

Coding

- The coding is the process of transforming the design of a system into a computer language format.

- This coding phase of software development is concerned with software translating design specification into the source code.

- It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.

- Coding is done by the coder or programmers who are independent people than the designer.

- The goal is not to reduce the effort and cost of the coding phase, but to cut to the cost of a later stage.

- The cost of testing and maintenance can be significantly reduced with efficient coding.

Goals of Coding

1. **To translate the design of system into a computer language format:** The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.

2. **To reduce the cost of later phases:** The cost of testing and maintenance can be significantly reduced with efficient coding.

3. **Making the program more readable:** Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.

Coding Standards

- General coding standards refers to how the developer writes code.

- Good software development organizations want their programmers to maintain to some well-defined and standard style of coding called coding standards.

- They usually make their own coding standards and guidelines depending on what suits their organization best and based on the types of software they develop.

- It is very important for the programmers to maintain the coding standards otherwise the code will be rejected during code review.

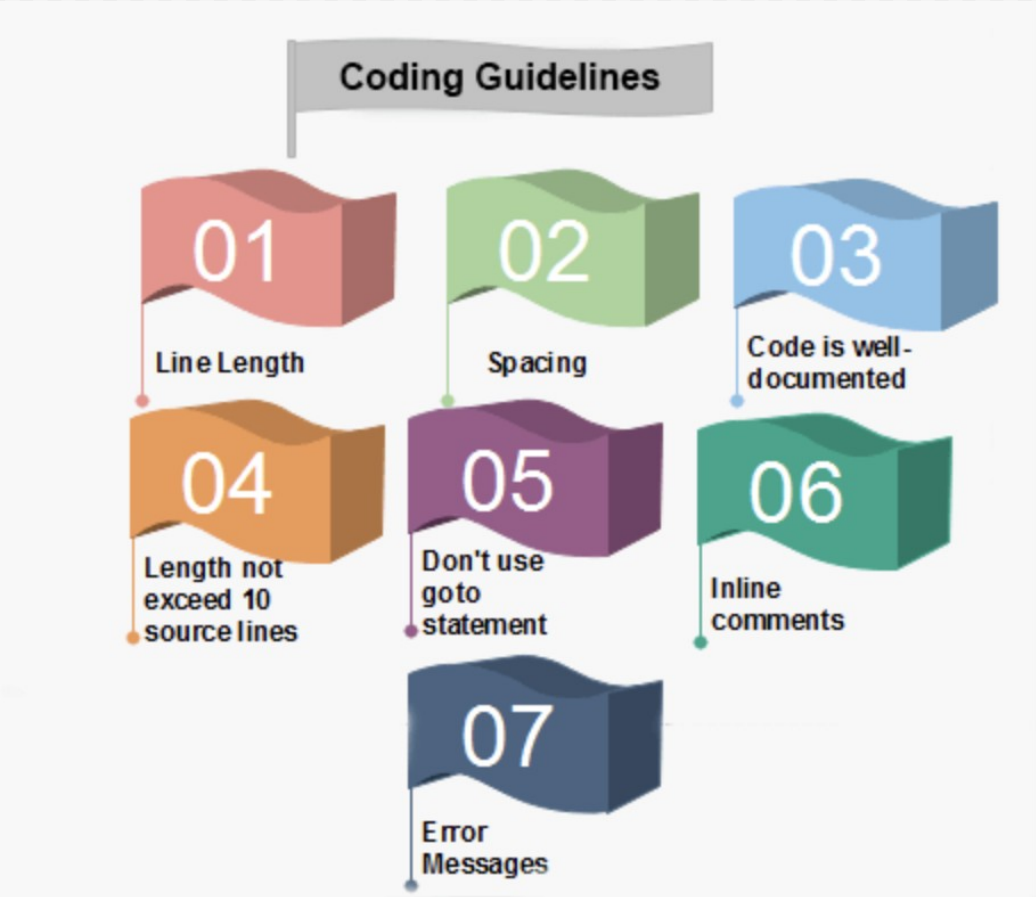
Purpose of Having Coding Standards

- A coding standard gives a uniform appearance to the codes written by different engineers.
- It improves readability, and maintainability of the code and it reduces complexity also.
- It helps in code reuse and helps to detect error easily.
- It promotes sound programming practices and increases efficiency of the programmers.

Coding Guidelines

- General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain.

- Most of the examples use the C language syntax, but the guidelines can be tested to all languages.



1. **Line Length:** It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

2. **Spacing:** The appropriate use of spaces within a line of code can improve readability.

Bad:

```
cost=price+(price*sales_tax)
fprintf(stdout ,"The total cost is %5.2f\n",cost);
```

Better:

```
cost = price + ( price * sales_tax )
fprintf (stdout,"The total cost is %5.2f\n",cost);
```

3. **The code should be well-documented:** As a rule of thumb, there must be at least one comment line on the average for every three-source line.

4. **The length of any function should not exceed 10 source lines:** A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.

5**. Do not use goto statements:** Use of goto statements makes a program unstructured and very tough to understand

6. **Inline Comments:** Inline comments promote readability.

7. **Error Messages**: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

Code Review

- Code review is a systematic and collaborative process in software development where one or more developers examine and evaluate another developer's code to identify any issues, provide any feedback, and ensure that the code meets quality standards.

- The primary goal is to improve the maintainability, quality, and reliability of the codebase resulting in the overall success of a project while also promoting knowledge sharing and learning among team members.

Why is Code Review important?

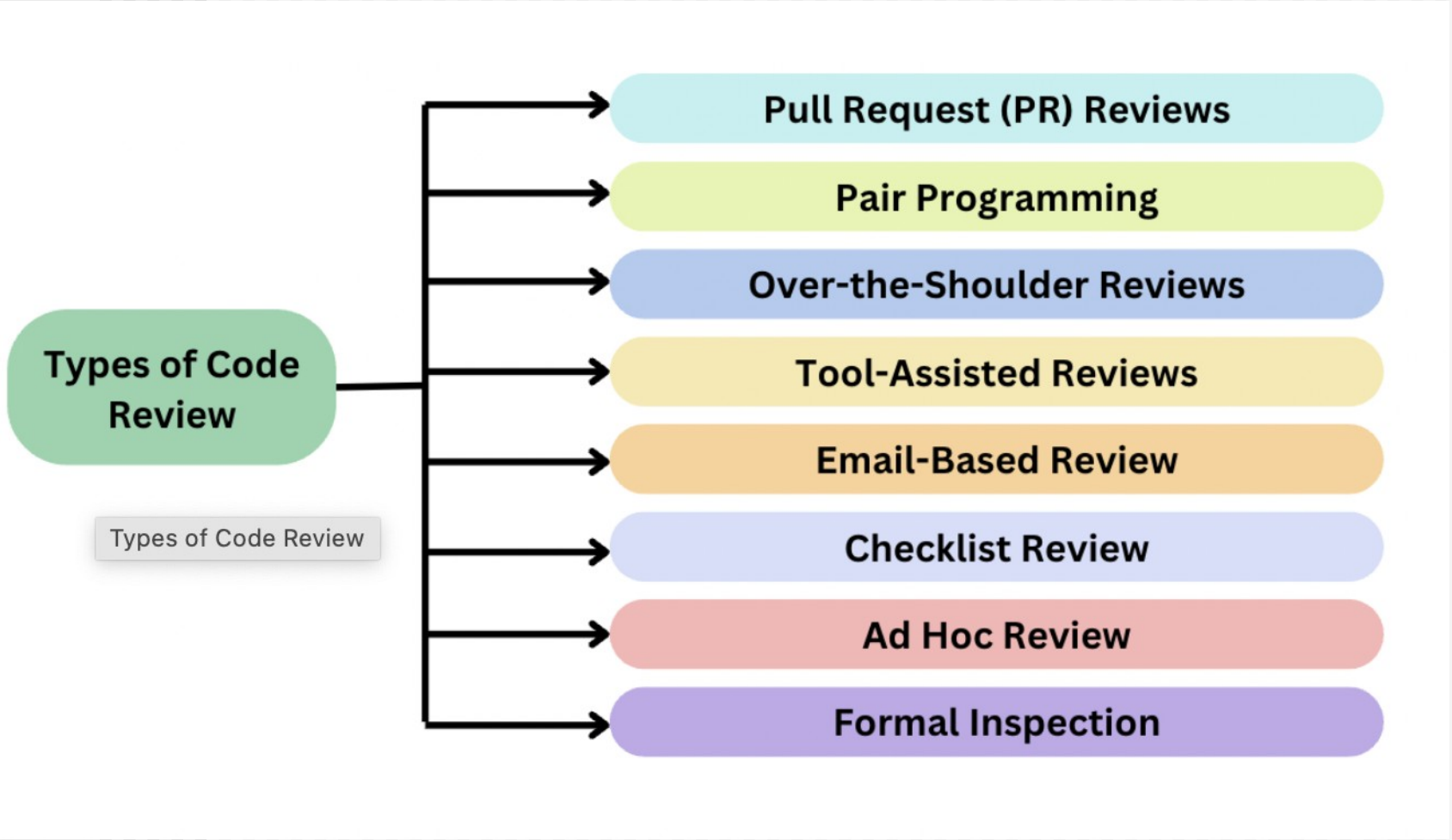
- **Bug Detection and Prevention:** Code review helps detect and address errors, bugs, and issues in the code before they propagate to production. This early detection significantly helps in reducing the cost and effort required to fix problems later in the development process or after deployment.

- **Improved Quality of Code:** Code review encourages writing of clean, maintainable, and efficient code. Reviewers can suggest enhancements, recommend best practices, and ensure that the code aligns with the established coding standards and guidelines.

- **Consistency:** Code review enforces consistency in coding style and practices across the team which makes the codebase easier to maintain and understand.

- **Knowledge Sharing and Learning:** Code reviews offer opportunities for knowledge sharing among team members to learn from one another. Discussions during these reviews can help spread domain knowledge and best practices.

Types of Code Review



- **Pull Request (PR) Reviews:** In Git-based version control systems like GitHub, etc., developers often create pull or merge requests to propose changes to the codebase. These pull requests are reviewed by the team members before the changes are merged.

- **Pair Programming:** In pair programming, two developers work together on the same computer, with one writing code while the other reviewing it in real time. This form of code review is highly interactive.

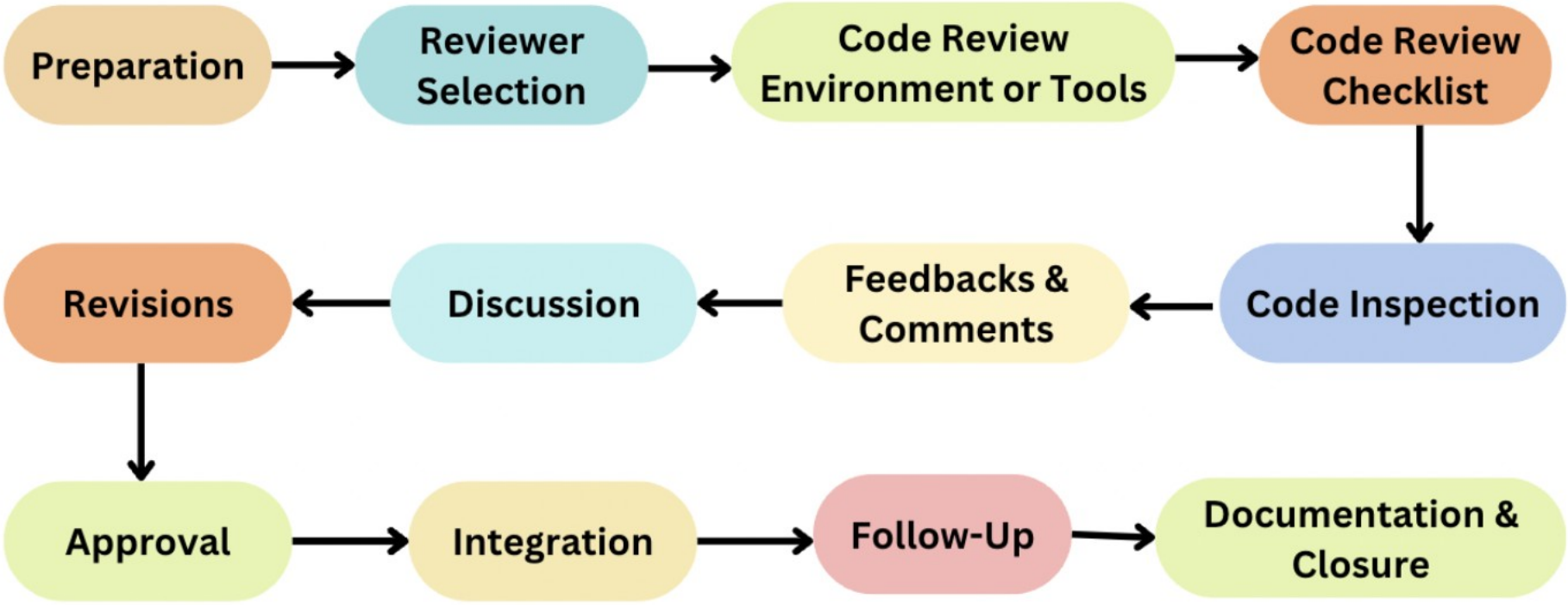
- **Over-the-Shoulder Reviews**: A developer may ask a team member to review their code by physically sitting together and going through the code on the computer screen.

- **Tool-Assisted Reviews**: Various tools and platforms are available to facilitate code reviews, such as GitHub, GitLab, Bitbucket, and code review-specific tools like Crucible and Review Board.

- **Email-Based Review:** In email-based reviews, code changes are sent via email, and the reviewers provide feedback and comments in response. The discussion occurs over email threads.

- **Checklist Review:** In a checklist review, a checklist is used to evaluate the code. Reviewers go through the list and check off items as they are reviewed.
- **Ad Hoc Review:** Ad hoc reviews are informal and spontaneous. Developers may spontaneously ask team members to take a quick look at their code or discuss changes without a formal process.
- **Formal Inspection:** Formal inspections are a structured form of code review that follows a predefined process. They often involve a dedicated inspection team and detailed documentation.

How to perform a Code Review?



- **Preparation:** The developer who has completed a piece of code or feature initiates the code review by creating a review request or notifying the team.
- **Reviewer Selection:** One or more team members are assigned as reviewers. Reviewers must have the relevant expertise to assess the code properly.
- **Code Review Environment or Tools:** Code reviews are often facilitated using specialized tools or integrated development environments (IDEs) that allow reviewers to view the code changes and leave feedback.
- **Code Review Checklist:** Reviewers may refer to a code review checklist or coding standards to ensure that the code follows established guidelines.
- **Code Inspection:** Reviewers examine the code thoroughly, looking for issues such as: syntax errors, logic errors, performance bottlenecks, security vulnerabilities, coding standard consistencies, adherence to best practices along with the code's overall design, scalability, and maintainability.

- **Feedbacks and Comments:** Reviewers provide comments within the code review tool, explaining any issues they find. These comments should be clear, constructive, and specific, making it easier for the author to understand the feedback.
- **Discussion:** The author and reviewers engage in a discussion about the code changes. This dialogue can involve clarifications, explanations, and suggestions for improvement.
- **Revisions:** The author makes necessary changes based on the feedback received and continues to engage in discussions until all concerns are addressed. This may involve multiple review iterations.
- **Approval:** Once reviewers are satisfied with the code changes and all concerns have been resolved, they approve the code for integration.
- **Integration:** The approved code changes are integrated into the main codebase, typically using version control systems.
- **Follow-Up:** After integration, it's crucial to monitor the code in the production environment to ensure that the changes work as expected and do not introduce new issues.
- **Documentation and Closure:** The code review process is documented for future reference, and the review is officially closed. This documentation may include the review comments, decisions made, and any action items.

Advantages

- **Enhanced Code Quality**: Code review helps identify and correct bugs early in the development process, resulting in higher code quality.
- **Knowledge Sharing:** Code reviews facilitate knowledge sharing among team members.
- **Consistency**: Code reviews enforce coding standards and guidelines, ensuring that the codebase is consistent and easier to maintain.
- **Security Improvements:** Code review can help identify and mitigate security vulnerabilities, improving the overall security of the software.
- **Team Collaboration:** Code reviews encourage collaboration and communication among team members. They provide a structured forum for discussing code changes and making collective decisions.
- **Code Reusability:** By examining code thoroughly, developers can identify opportunities for code reuse, leading to more efficient and maintainable software development.

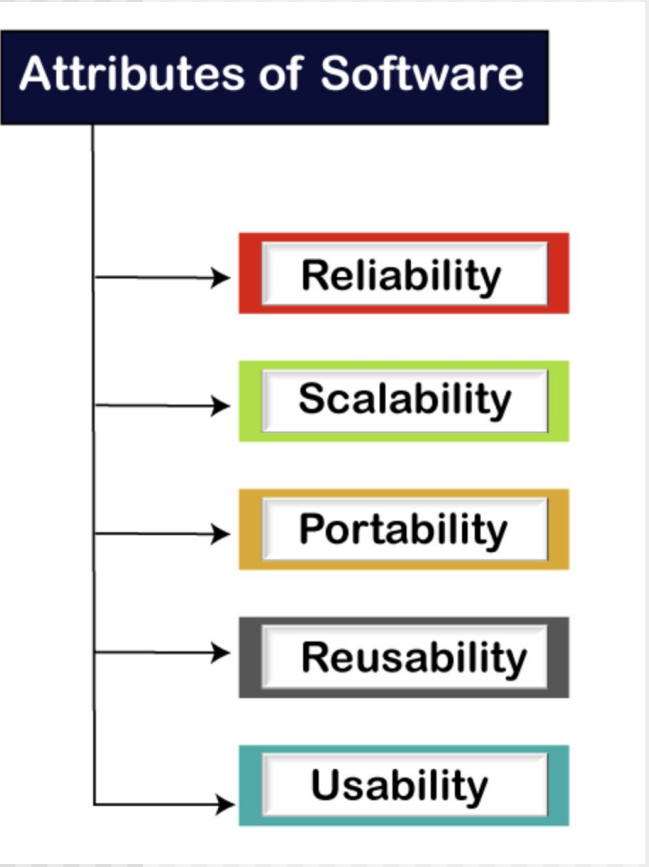
Limitations

- **Time-Consuming:** Code review can be time-consuming, especially for large or complex code changes. This may lead to delays in the development process.
- **Subjectivity:** Code review feedback can be subjective, and different reviewers may have different opinions on the code style and quality.
- **Overhead:** Excessive code reviews, especially for trivial changes, can create unnecessary overhead and slow down development.
- **Reviewer Bias**: Reviewers may have biases or blind spots that can influence their feedback. They may focus on certain aspects of the code while overlooking others.
- **Communication Challenges:** Reviewers and authors may face challenges in effectively communicating feedback and understanding each other's perspectives.

Software Testing

- Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability).

- Evaluating the execution of software components to find the software bugs or errors or defects.



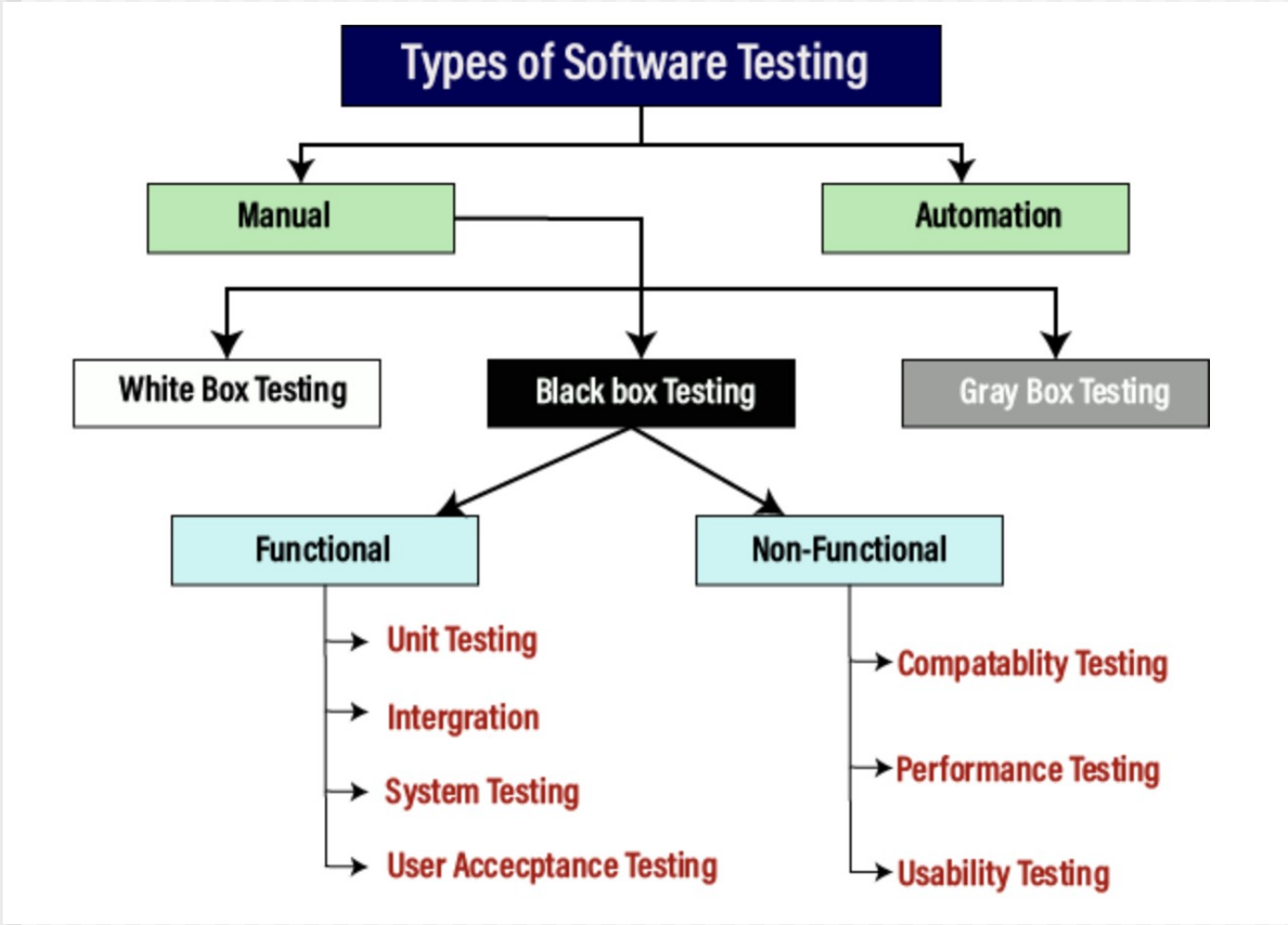
- Software testing provides an independent view and objective of the software and gives surety of fitness of the software.

- It involves testing of all components under the required services to confirm that whether it is satisfying the specified requirements or not.

- The process is also providing the client with information about the quality of the software.

- Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing.

- So, without testing software cannot be deployed to the end user.


Types of Software Testing
Manual
Automation
White Box Testing
Black box Testing
Gray Box Testing
Functional
Non-Functional
Unit Testing
Intergration
System Testing
User Accecptance Testing
Compatablity Testing
Performance Testing
Usability 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,

1. White Box Testing
2. Black Box Testing
3. 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.

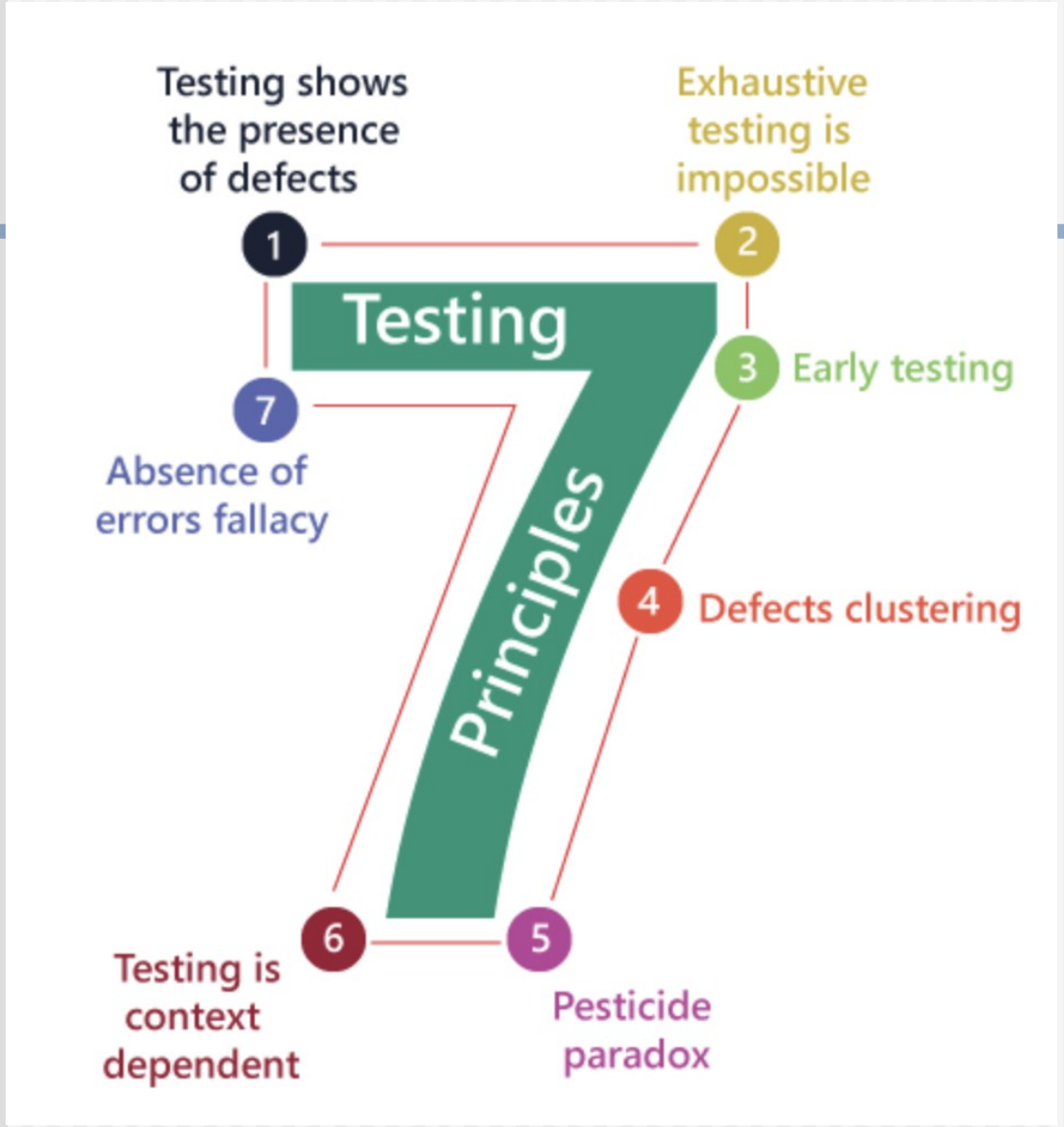
- We need to write a test script and execute those scripts.

Software Testing Principles

- Software testing is a procedure 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.

1. Testing shows the presence of defects
2. Exhaustive Testing is not possible
3. Early Testing
4. Defect Clustering
5. Pesticide Paradox
6. Testing is context-dependent
7. Absence of errors fallacy


Testing shows the presence of defects
Exhaustive testing is impossible
1
2
Testing
7 Principles
3 Early testing
7
Absence of errors fallacy
4 Defects clustering
6
5
Testing is context dependent
Pesticide paradox

1. Testing shows the presence of defects

- 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.

Testing is not about proving that software is error-free but about finding defects that need to be fixed. Defects can include code bugs, missing requirements, incorrect functionality, and other issues.

2. Exhaustive testing is time consuming

- 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.
- Instead of performing the exhaustive testing as it takes boundless determinations and most of the hard work is unsuccessful.
- We can complete this type of variations according to the importance of the modules because the product timelines will not permit us to perform such type of testing scenarios.

It is impossible to test all possible combinations and scenarios in software applications, so testers must focus on the most critical and high-risk areas.

3. Early Testing saves time and money

- Early testing means that all the testing activities should start in the early stages of the software development life cycle's to identify the defects .

- If we find the bugs at an early stage, it will be fixed in the initial stage itself

- 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.

- 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.

It is more cost-effective to identify and fix defects early in the software development life cycle than later in the process.

4. 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 various reasons 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.

- 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.

In software development, defects often occur in clusters, meaning that a small number of modules or components are responsible for the majority of defects. Testers should focus on these high-risk areas.

5. 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.

Repeating the same tests with the same inputs can result in the same defects being identified repeatedly. Testers must continually update and modify their test cases to find new defects.

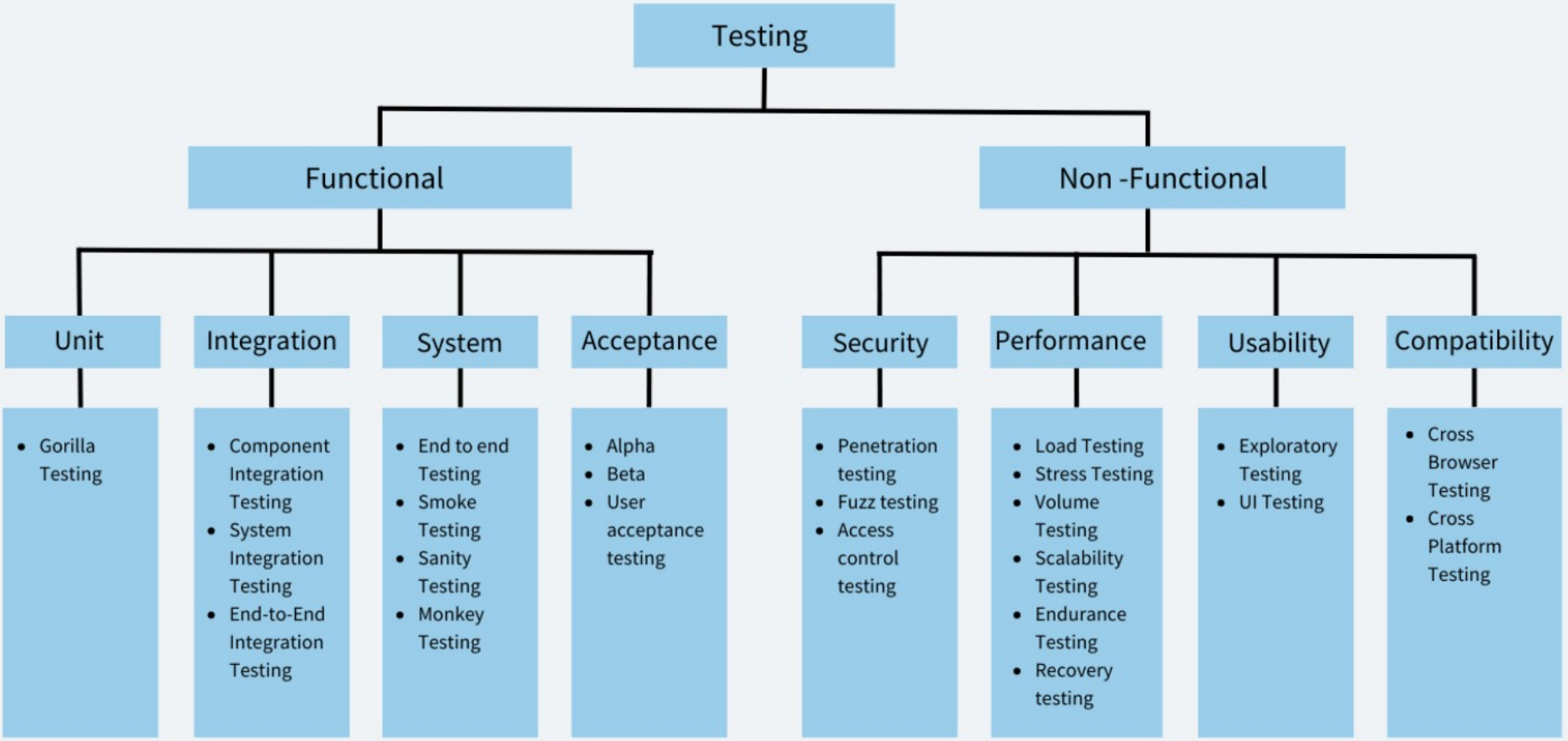
6. Testing is context dependent

- 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.

The testing process should be tailored to the specific context and requirements of the software application being tested.

7. Absence of error fallacy

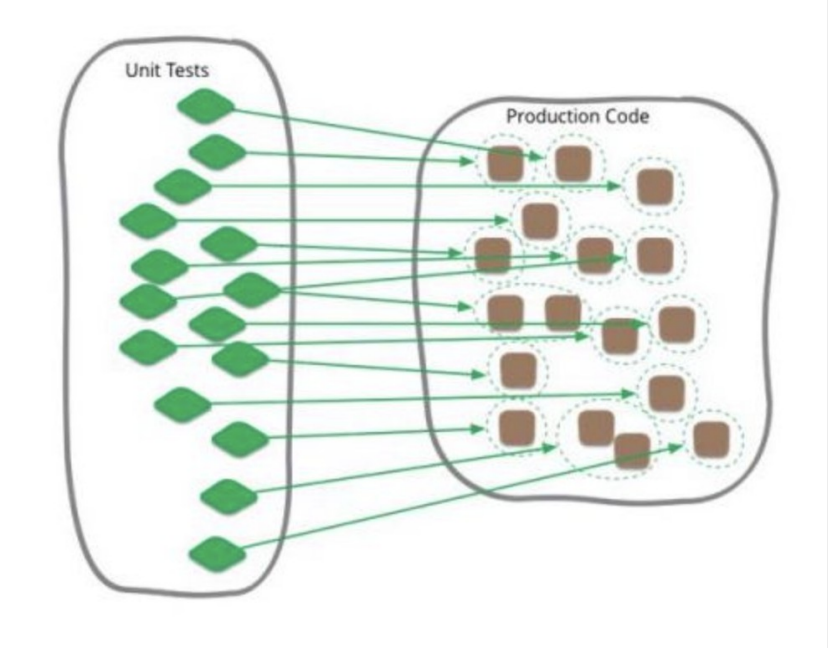
- Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free.
- But there is the chance when the application is tested beside the incorrect requirements, identified the flaws, and fixed them on a given period 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.


Testing
Functional
Non -Functional
Unit
Integration
System
Acceptance
Security
Performance
Usability
Compatibility
Gorilla Testing
Component Integration Testing
System Integration Testing
End-to-End Integration Testing
End to end Testing
Smoke Testing
Sanity Testing
Monkey Testing
Alpha
Beta
User acceptance testing
Penetration testing
Fuzz testing
Access control testing
Load Testing
Stress Testing
Volume Testing
Scalability Testing
Endurance Testing
Recovery testing
Exploratory Testing
UI Testing
Cross Browser Testing
Cross Platform Testing

Types of Functional Testing

1. Unit Testing

- Unit testing is a software testing type in which individual units/components are tested in isolation from the rest of the system to ensure that they work as intended.
- A unit refers to the smallest testable part of a software application that performs a specific function or behavior.
- A unit can be a method, a function, a class, or even a module.
- They can be run in isolation or in groups. Unit tests are typically written by developers to check the correctness of their code and ensure that it meets the requirements and specifications.



Example

A developer has scripted a password input text field with its validation (8 characters long, must contain special characters.); makes a unit test to test out this one specific text field. (has a test that only inputs 7 characters, no special characters, empty field)

Advantages of Unit Testing

- Early detection of Bugs
- Simplifies Debugging Process
- Encourages Code Reusability
- Improves Code Quality
- Enables Continuous Integration

Types of Unit Testing

Gorilla Testing

- Gorilla testing is a software testing technique where the tester performs testing of a particular module or component of the software system rigorously and extensively to identify any issues or bugs that may arise.

- It focuses on testing a single module or component in depth to ensure that it can handle high loads and perform optimally under extreme conditions.

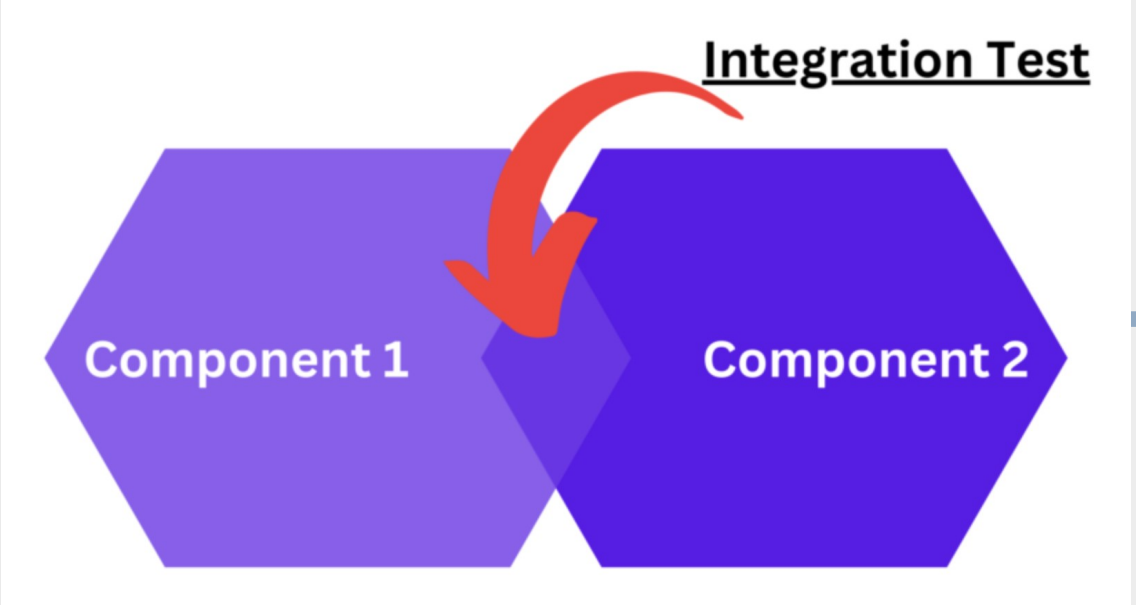
Advantages of Gorilla Testing

- Identify potential bottlenecks or weaknesses in a particular module
- Capable of handling high loads
- Helps identify issues or bugs that may be missed by other testing techniques

2. Integration Testing

- Integration testing is a testing type in which different modules or components of a software application are tested together as a group to ensure that they work as intended and are integrated correctly.

- The main aim of integration tests is to identify issues that might come up when multiple components work together. It ensures that individual code units/ pieces can work as a whole cohesively.



Integration testing can be further broken down to:

- **Component Integration Testing**: This type of testing focuses on testing the interactions between individual components or modules.
- **System Integration Testing**: Focuses on testing the interactions between different subsystems or layers of the software application.
- **End-to-End Integration Testing**: This type of integration testing focuses on testing the interactions between the entire software application and any external systems it depends on.

Example of Integration Tests

A software application consists of a web-based front-end, a middleware layer that processes data, and a back-end database that stores data. Integration tests would verify if the data submitted in the front end is processed by the middleware and then stored by the backend database.

Advantages of Integration testing

- Early Detection of Issues
- Improved Software Quality
- Increased Confidence in the Software
- Reduced Risk of Bugs in Production
- Better Collaboration Among Team Members
- More Accurate Estimation of Project Timelines

3. System Testing

- System testing is a testing type that tests the entire software application as a whole and ensures that the software meets its functional and non-functional requirements.

- System testing is typically performed after integration testing. During system testing, testers evaluate the software application's behavior in various scenarios and under different conditions, including normal and abnormal usage, to ensure that it can handle different situations effectively.

Example of System Testing

A software application consists of a web-based front-end, a middleware layer that processes data, and a back-end database that stores data. The system test for this scenario would involve the following steps

- The user accesses the front-end interface and submits an order, including item details and shipping information.
- The middleware layer receives the order and processes it, including verifying that the order is valid and the inventory is available.
- The middleware layer sends the order information to the back-end database, which stores the information and sends a confirmation message back to the middleware layer.
- The middleware layer receives the confirmation message and sends a response back to the front-end indicating that the order has been successfully processed.

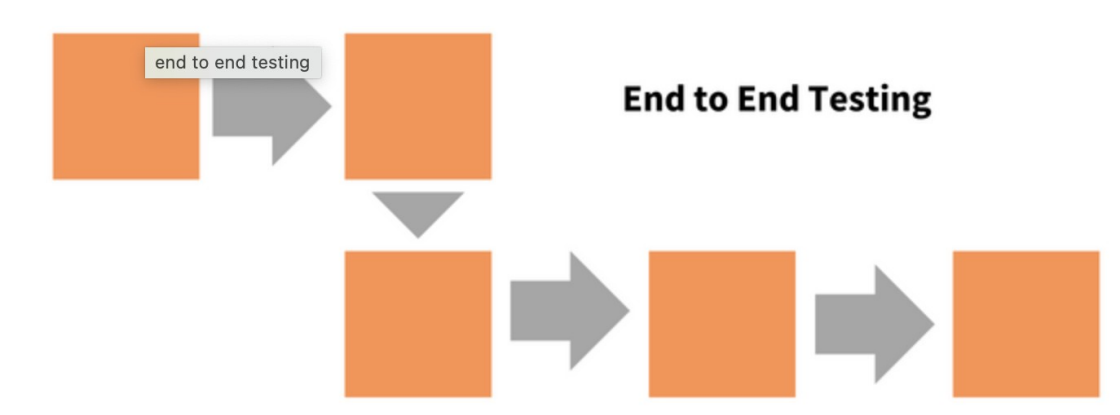
Advantages of System Testing

- Identifies and resolves defects or issues that may have been missed during earlier stages of testing.
- Evaluates the software application's overall quality, including its reliability, maintainability, and scalability.
- Increases user satisfaction
- Reduces risk

Types of System Testing

a. End to End Testing

- End-to-end testing is a testing methodology that tests the entire software system from start to finish, simulating a real-world user scenario.
- The goal of end-to-end testing is to ensure that all the components work together seamlessly and meet the desired business requirements.
- Most often people use the term system testing and end to end testing interchangeably. However both of them are different types of testing.
- System testing is a type of testing that verifies the entire system or software application is working correctly as a whole.
- This type of testing includes testing all the modules, components, and integrations of the software system to ensure that they are working together correctly.
- The focus of system testing is to check the system's behavior as a whole and verify that it meets the business requirements.



- End-to-end testing is a type of testing that verifies the entire software application from start to finish, including all the systems, components, and integrations involved in the application's workflow.

- The focus of E2E testing is on the business processes and user scenarios to ensure that they are working correctly and meet the user requirements.

Example of End to End Testing

E-commerce transaction: End-to-end testing of an e-commerce website involves testing the entire user journey, from product selection to payment, shipping, and order confirmation.

b. Monkey Testing

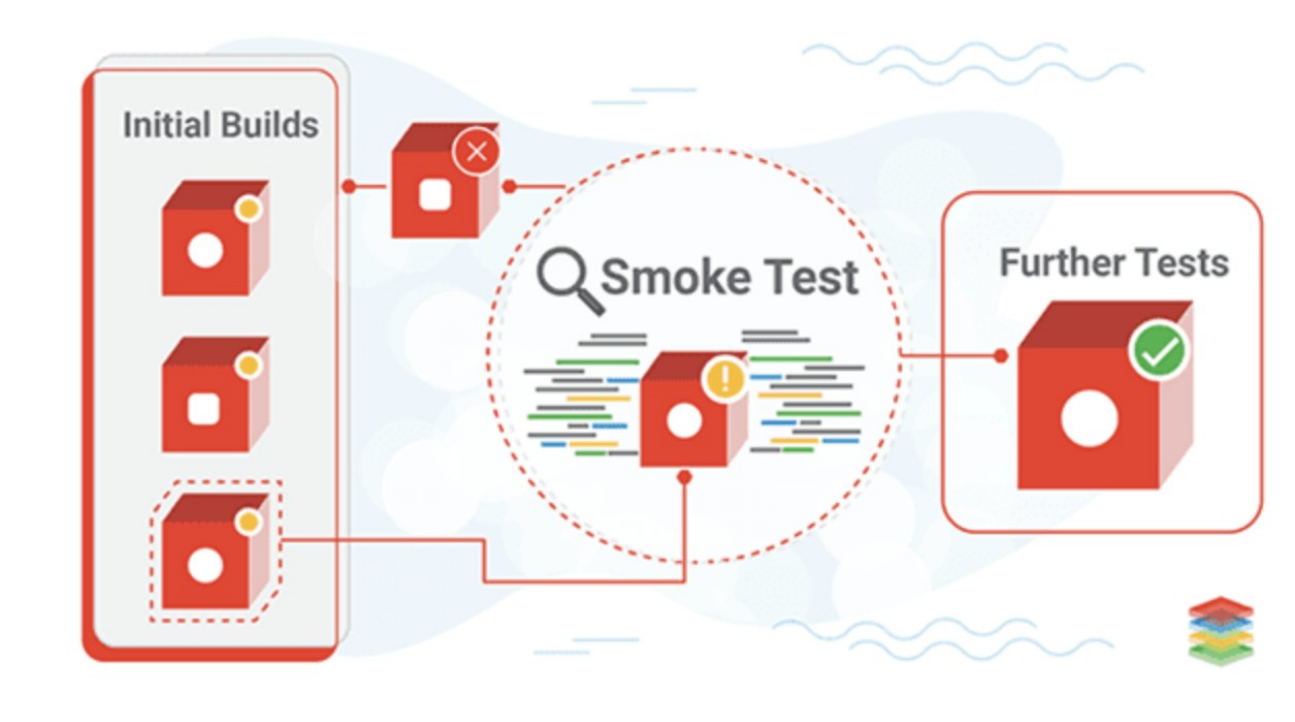
- Monkey testing is a testing type where the tester tests in a random manner with random inputs to analyze if the application breaks.
- The objective of monkey testing is to verify if an application crashes by giving random input values.
- There are no special test cases written for monkey testing.

Example of Monkey Testing

A tester randomly turning off the power or unplugs the system to test the application's ability to recover from sudden power failures.

c. Smoke Testing

Smoke testing is a testing type that is conducted to ensure that the basic and essential functionalities of an application or system are working as expected before moving on to more in-depth testing.



4. Acceptance Testing

- Acceptance Testing verifies whether a software application meets the specified acceptance criteria and is ready for deployment.
- It is usually performed by end-users or stakeholders to ensure that the software meets their requirements and is fit for purpose.
- Acceptance Testing can be further divided into two types:

  1. User Acceptance Testing (UAT)
  2. Business Acceptance Testing (BAT).

- User Acceptance Testing is performed by end-users to validate that the software meets their needs and is easy to use.
- Business Acceptance Testing is performed by stakeholders to ensure the alignment of business/functional requirements with the organization's objectives.

Example of Acceptance Testing

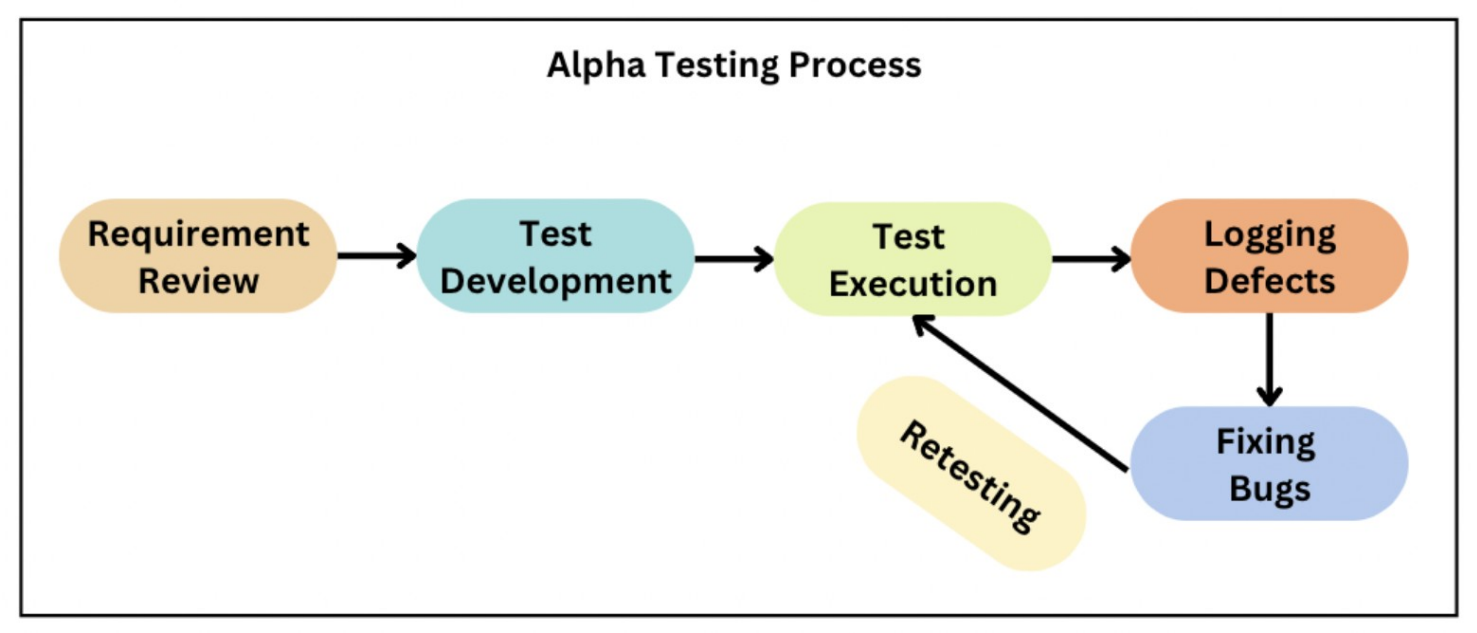
Conducting tests to meet if an app meets the requirements of the user.

For a banking app, acceptance testing would involve testing the app for login, account management, fund transfer, statement download, card payment etc.

Types of Acceptance Testing

a. Alpha Testing

Alpha testing is a type of testing that is performed in-house by the development team or a small group of users. It is the first phase of testing that is conducted before the software is released to the public or external users. Alpha testing is a crucial step in the software development process as it helps to identify bugs, defects, and usability issues before the product is released.



Example of Alpha Testing

A game development company is creating a new game. The development team performs alpha testing by testing the game's performance, such as loading times, graphics, sound effects, and gameplay.

Advantages of Alpha Testing

- Early detection of issues
- Enhanced user experience
- Feedback from internal users

b. Beta Testing

- Beta testing is a type of testing that is performed by a group of external users who are not a part of the development team.
- The purpose of beta testing is to gather feedback from real users and to identify any issues that were not found during the alpha testing phase.

Example of Beta Testing

A software company is releasing a new feature of its product. The company invites a group of external users to beta test the product and provide feedback on any bugs, defects, or issues that were not found during the alpha testing phase.

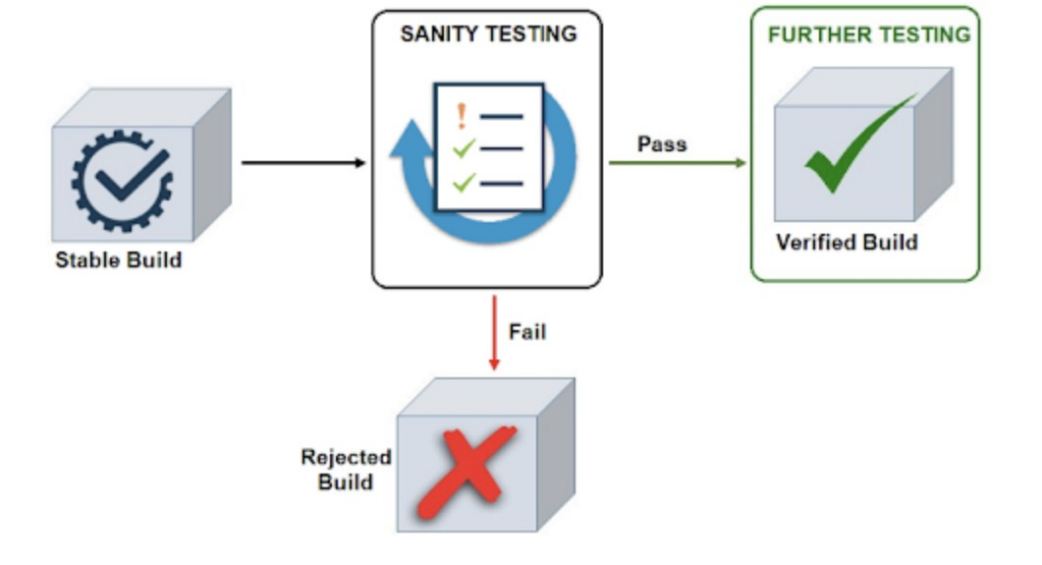
Advantages of Beta Testing

- Real-world feedback
- Marketing and promotion
- Enhanced user experience

c. Sanity Testing

Sanity testing is a testing type that is performed quickly determine if a particular functionality or a small section of the software is working as expected after making minor changes.

The main objective of sanity testing is to ensure the stability of the software system and to check whether the software is ready for more comprehensive testing.



Example of Sanity Testing

After fixing a bug that caused a crash in a mobile application, you can perform sanity testing by opening the app and ensuring that it does not crash anymore.

Advantages of Sanity Testing

- Quick and efficient
- Saves time and effort
- Cost-effective

Types of Non Functional Testing

1. Security Testing

Security testing is a type of software testing that assesses the security of a software application. It helps to identify vulnerabilities and weaknesses in the system and ensure that sensitive data is protected.

Examples of security testing include penetration testing, vulnerability scanning, and authentication testing.

Advantages of Security Testing

- Improved system security
- Protection of sensitive data
- Compliance with regulations

Types of Security Testing

- **Penetration testing**: This involves attempting to exploit potential vulnerabilities in the software system by simulating an attack from a hacker or other malicious actor.
- **Fuzz testing:** This involves sending many unexpected or malformed input data to the software system to identify potential vulnerabilities related to input validation and handling.

2. Performance Testing

- Performance testing is a type of software testing that assesses the performance and response time of a software application under different workloads.
- It helps to identify bottlenecks in the system and improve the performance of the application.

Examples of performance testing include load testing, stress testing, and volume testing.

Advantages of Performance Testing

- Increased customer satisfaction
- Better scalability
- Improved user experience

Types of Performance Testing

a. Load Testing

- Load testing is a type of performance testing that assesses the performance and response time of a software application under a specific workload.
- It helps to identify the maximum capacity of the system and ensure that it can handle the expected user load.

Examples of Load Testing

Simulating multiple users accessing a website at the same time or performing multiple transactions on a database.

Advantages of Load Testing

- Improved system reliability
- Better scalability

b. Stress Testing

- Stress testing is a type of performance testing that assesses the performance and response time of a software application under extreme workloads.
- It helps to identify the system's breaking point and ensure that it can handle unexpected workloads.

Examples of Stress Testing

Simulating thousands of users accessing a website simultaneously or performing millions of transactions on a database.

Advantages of Stress Testing

- Improved system reliability
- Better preparedness for real-world scenarios
- Better scalability

c. Volume Testing

- Volume testing is a type of testing that assesses the performance and response time of a software application under a specific volume of data.

- It helps to identify the system's capacity to handle large volumes of data.

Examples of Volume Testing

Inserting large amounts of data into a database or generating large amounts of traffic to a website.

Advantages of Volume Testing

- Improved system reliability
- Better scalability

d. Scalability Testing

- Scalability testing evaluates the software's ability to handle increasing workload and scale up or down in response to changing user demands.

- It involves testing the software system under a range of different load conditions to determine how it performs and whether it can handle increasing levels of traffic, data, or transactions.

Examples of Scalability Testing

Testing a website by gradually increasing the number of simulated users accessing the website and tracking how the system responds to it.

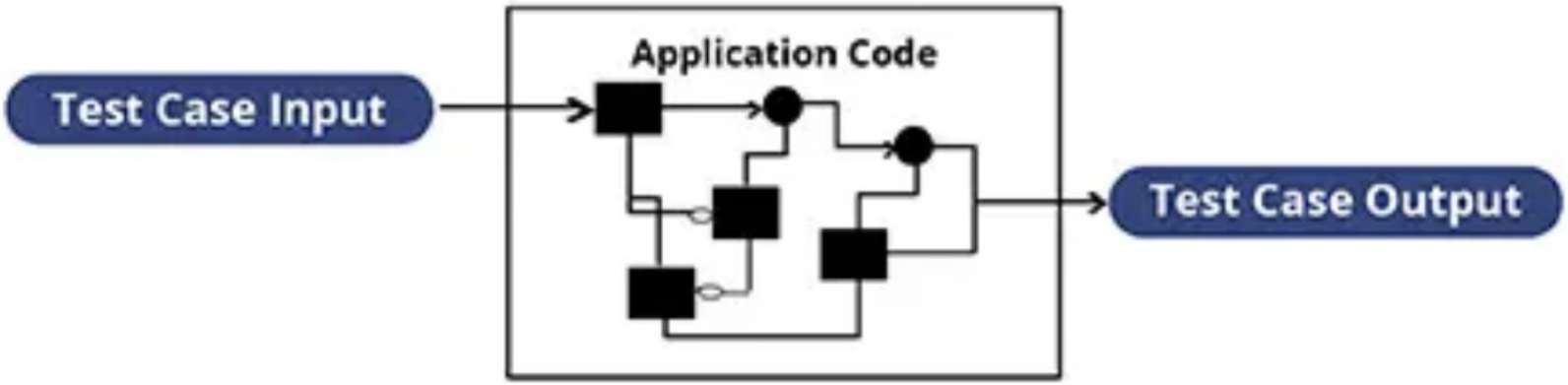
Advantages of Scalability Testing

- Optimize system performance
- Better scalability

White Box Testing

- White Box Testing, or structural, code-based, or glass box testing, is a software testing technique that focuses on the software's internal logic, structure, and coding.
- It provides testers with complete application knowledge, including access to source code and design documents, enabling them to inspect and verify the software's inner workings, infrastructure, and integrations.
- Test cases are designed using an internal system perspective, and the methodology assumes explicit knowledge of the software's internal structure and implementation details.
- This in-depth visibility allows White Box Testing to identify issues that may be invisible to other testing methods.

WHITE BOX TESTING APPROACH



White Box Testing is a more exhaustive and detailed testing method that provides a deep understanding of the software's internal structure, making it a vital part of the software testing process.

The key benefits of White Box Testing include,

- **Thoroughness**: It provides complete code coverage, ensuring every part of the software's internal structure is tested.
- **Automation**: Test cases can be easily automated, saving time and resources.
- **Optimization**: It helps in code optimization by identifying hidden errors and redundancies.
- **Introspection**: It provides an in-depth understanding of the software, which can be invaluable for future development and maintenance.

Code coverage

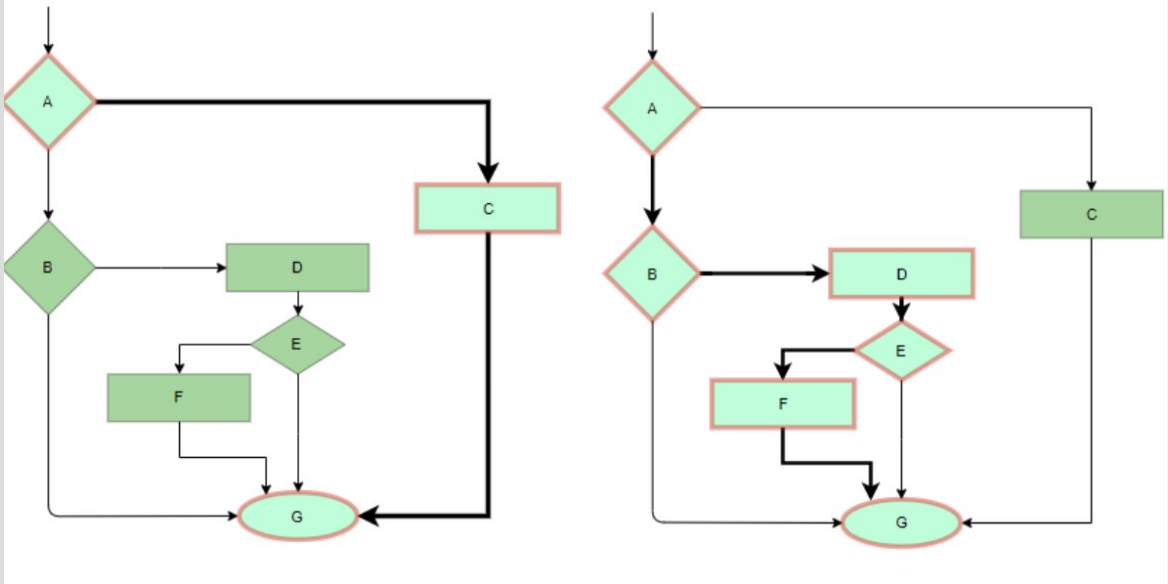
- Code coverage is a measure which describes the degree of which the source code of the program has been tested.
- It is one form of white box testing which finds the areas of the program not exercised by a set of test cases.
- It also creates some test cases to increase coverage and determining a quantitative measure of code coverage.

Testing Techniques

- Statement coverage
- Branch coverage
- Path coverage
- Loop coverage
- Condition testing

1. Statement Coverage

In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.



- Imagine you're a teacher checking a student's homework, Statement coverage would be like ensuring the student has answered every question on the assignment

- Statement Coverage is a white box testing technique in which all the executable statements in the source code are executed at least once.

- It is used for calculation of the number of statements in source code which have been executed.

- The main purpose of Statement Coverage is to cover all the possible paths, lines and statements in source code.

- Statement coverage is used to derive scenario based upon the structure of the code under test.

$$\text{Statement Coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

Example

```
Prints (int a, int b)
int result = a + b;
     If (result> 0)
     Print ("Positive", result)
     Else
     Print ("Negative", result)
     }
```

Scenario 1

If A = 5, B = 2

```
Prints (int a, int b)
int result = a+ b;
     If (result> 0)
      Print ("Positive", result)
     Else
      Print ("Negative", result)
     }
```

The statements marked in pink colour are those which are executed as per the scenario

Number of executed statements = 5, Total number of statements = 7

Statement Coverage: 5/7 = 71%

Scenario 2

If A = -5, B = -2

```
Prints (int a, int b)
int result = a+ b;
     If (result> 0)
      Print ("Positive", result)
     Else
      Print ("Negative", result)
     }
```

The statements marked in pink colour are those which are executed as per the scenario

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

Statement Coverage:6/7 = 85%

Decision Coverage Testing

- Decision Coverage is a white box testing technique which reports the true or false outcomes of each boolean expression of the source code.

- The goal of decision coverage testing is to cover and validate all the accessible source code by checking and ensuring that each branch of every possible decision point is executed at least once.

$$Decision\ Coverage = \frac{Number\ of\ Decision\ Outcomes\ Excercised}{Total\ Number\ of\ Decision\ Outcomes}$$

Example

```
Demo(int a) {
      If (a> 5)
       a=a*3
      Print (a)
     }
```

Scenario 1

Value of a is 2

```
Demo(int a) {
      If (a> 5)
      a=a*3
      Print (a)
     }
```

The code highlighted will be executed.

Here the outcome of the decision If (a>5) is checked is “No”.

Decision Coverage = 50%

Example

```
Demo(int a) {
      If (a> 5)
       a=a*3
      Print (a)
     }
```

Scenario 1

Value of a is 7

```
Demo(int a) {
      If (a> 5)
       a=a*3
      Print (a)
     }
```

The code highlighted will be executed.

Here the outcome of the decision If (a>5) is checked is “Yes”.

Decision Coverage = 50%

2. Branch Coverage

In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.

This is like exploring all possible routes on a GPS. If you're at an intersection, branch testing involves going straight, turning left, and turning right to ensure all paths lead to valid destinations.

- Branch Coverage is a white box testing method in which every outcome from a code module(statement or loop) is tested.

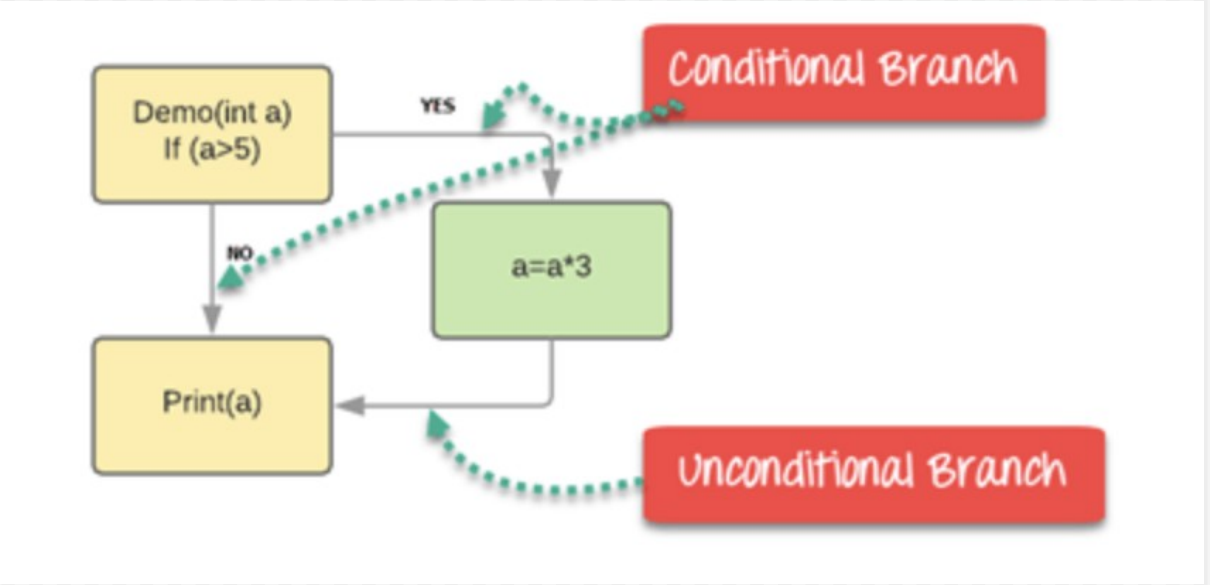
- The purpose of branch coverage is to ensure that each decision condition from every branch is executed at least once.

- It helps to measure fractions of independent code segments and to find out sections having no branches.

$$Branch\ Coverage = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches}$$

Example

```
Demo(int a) {
      If (a> 5)
      a=a*3
      Print (a)
    }
```

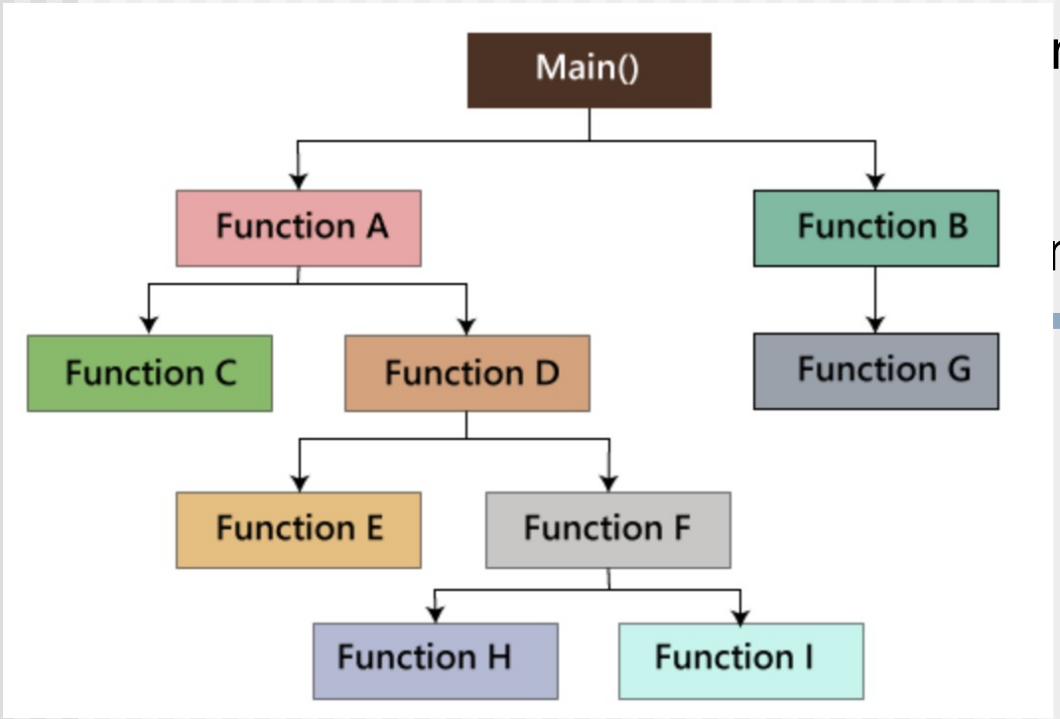


Test Case	Value of A	Output	Decision Coverage	Branch Coverage
1	2	2	50%	33%
2	6	18	50%	67%

Advantages

- Allows you to validate-all the branches in the code
- Helps you to ensure that no branched lead to any abnormality of the program's operation
- Branch coverage method removes issues which happen because of statement coverage testing
- Allows you to find those areas which are not tested by other testing methods
- It allows you to find a quantitative measure of code coverage
- Branch coverage ignores branches inside the Boolean expressions

3. Path coverage



ndependent paths.

graphs are representing the every program is added with mage:

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 the same way test all other paths and fix the bugs.

- This would be like a postman ensuring they can deliver mail to every house on their route. They need to make sure every possible path is covered.

4. Loop coverage

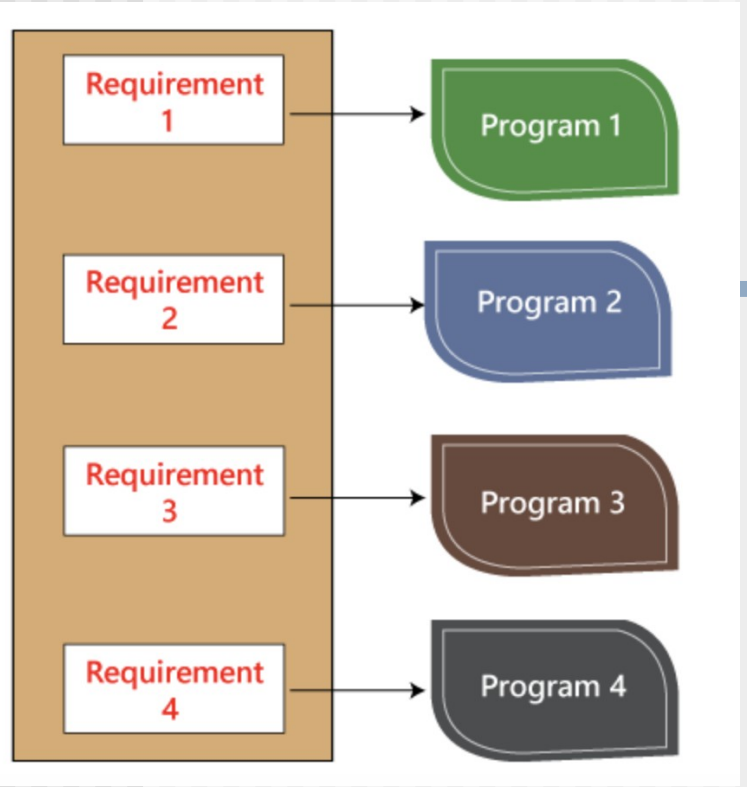
- We will test the loops such as while, for, and do-while, etc. and also check for ending condition if working correctly and if the size of the conditions is enough.

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

- 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.

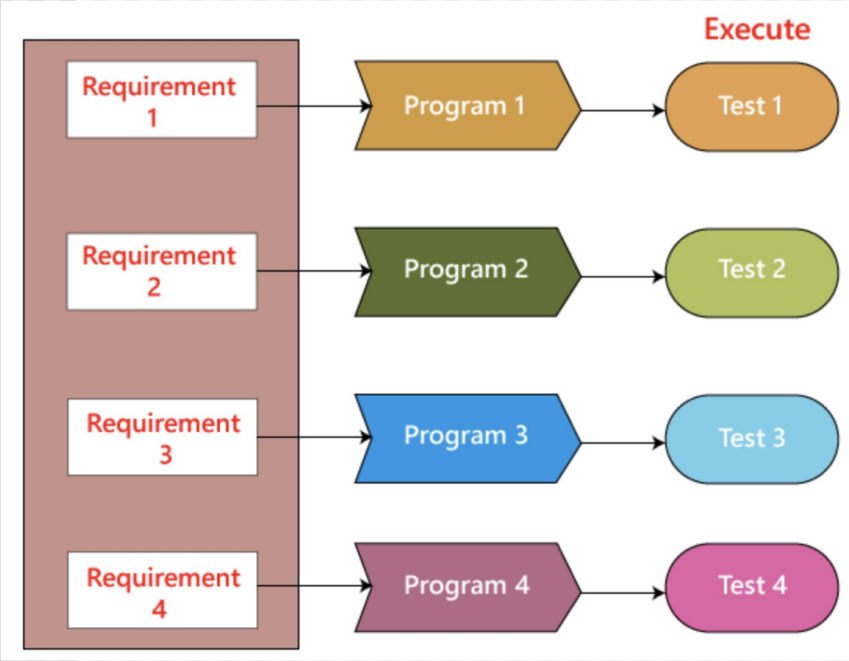
- It is written by the developers only.



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 take lots of time and efforts.



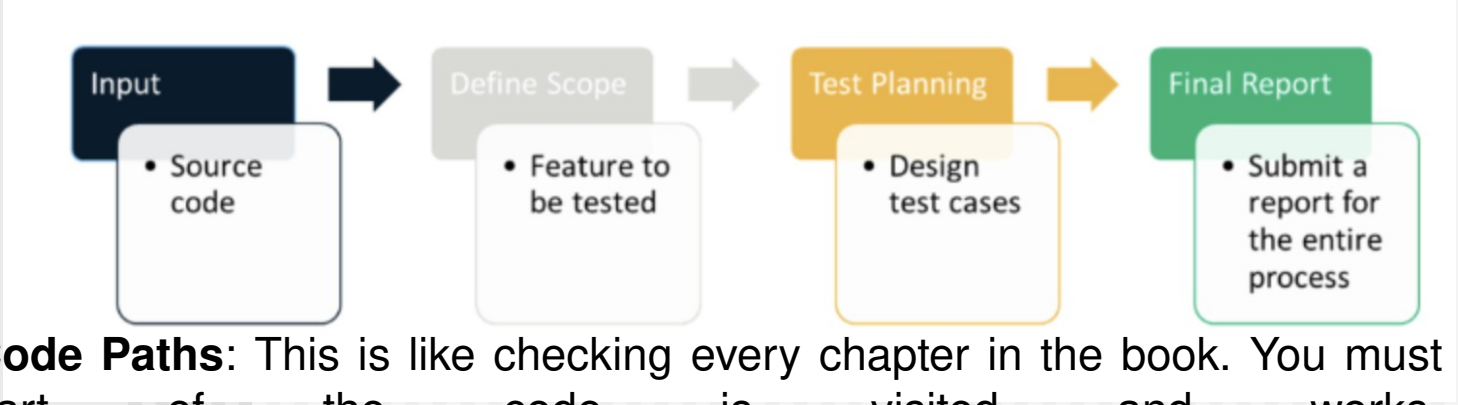
solved in the following ways:

ar program where the developer writes guage as the source code. Then they so known as **unit test programs**. These rogram 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 then executes the test program.

- This is like checking a playlist on repeat. You want to ensure it loops back to the first song correctly after the last song finishes.

What to Verify in White Box Testing?



- **Code Paths**: This is like checking every chapter in the book. You must ensure every part of the code is visited and works correctly.
- **Loops**: This is like checking if the book has any repeated chapters. In code, loops are parts that can run multiple times. You must ensure these loops work correctly and don't repeat forever.
- **Conditions**: This is like checking if the book has twists and turns. In code, conditions can change what the software does. You need to ensure that every possible outcome of these conditions is tested.
- **Inputs and Outputs**: This is like checking the start and end of the book. You need to ensure that for every input (start), the software produces the correct output (end).
- **Individual Parts**: This is like checking every character in the book. In code, these are the individual functions or objects. You need to make sure each one works correctly on its own.

Advantages	Limitations
Thorough Testing: Like reading every chapter of a book, White Box Testing checks every part of the code, making it very thorough.	**Complexity:** Just like a book can be hard to understand if it's written in a difficult language, White Box Testing can be complex because it requires understanding the code.
Early Bug Detection: Like spotting a typo in the first few pages of a book, White Box Testing can find bugs early in the development process, which makes them cheaper and easier to fix.	**Time-Consuming:** Just like reading a long book can take a lot of time, White Box Testing can be time-consuming because it's so thorough.
Improves Security: Like a book review that warns about inappropriate content, White Box Testing can find security issues in the code, helping to make the software more secure.	**Requires Expertise:** Just like understanding a book written in an old or foreign language requires special knowledge, White Box Testing requires a deep understanding of coding and implementation.
Optimizes Code: Like a book review that suggests removing unnecessary chapters, White Box Testing can find unnecessary or redundant code, helping to make the software more efficient.	**Bias:** Just like a book reviewer might miss flaws in a book they love, developers who test their own code might miss bugs because they're too familiar with it.

Black Box Testing

- Black box testing is a software testing technique where the internal workings or code structure of the system being tested are not known to the tester.
- In other words, the tester focuses solely on the external behaviour of the software, without having access to its internal source code. The name “black box” comes from the idea that the internal workings are hidden or “boxed” from the tester’s view.

- **Independent Testing**: Black box testing is typically performed by testers who are independent of the development team. This ensures a fresh perspective and helps identify issues that developers might overlook.

- **Requirements-Based Testing**: Testers design test cases based on the software's requirements and specifications, without being concerned about how the code is implemented.

- **Functional Testing**: The main goal of black box testing is to assess the functionality of the software, checking if it meets the expected behaviour and delivers the desired outputs for various inputs.

- **No Knowledge of Internal Code**: Testers do not have access to the source code, architecture, or design details of the software. They interact with the system through its user interfaces or APIs.

Different Types of Black Box Testing

- Due to time and budget considerations, it is not possible to perform exhausting testing for each set of test data, especially when there is a large pool of input combinations.
- We need an easy way or special techniques that can select test cases intelligently from the pool of test-case, such that all test scenarios are covered.
- We use two techniques – **Equivalence Partitioning & Boundary Value Analysis testing techniques.**
- **Boundary Value Analysis (BVA):** BVA is a technique used to identify defects around the boundaries of input values. Test cases are designed with values at the edges of input ranges to assess how the software handles minimum and maximum limits.
- **Equivalence Partitioning:** In this technique, the input domain is divided into groups of data that are expected to behave similarly. Test cases are then derived from these partitions to minimise redundant testing.

Boundary Value Analysis (BVA)

- Boundary Value Analysis (BVA) is a Black-Box testing technique used to check the errors at the boundaries of an input domain.

- The name comes from the Boundary, which means the limits of an area. So, BVA mainly focuses on testing both valid and invalid input parameters for a given range of a software component.

- If (Min,MAX) is the range given for a field validation, then the boundary values come as follows:

Invalid Boundary Check { Min-1 ; Max+1 }
Valid Boundary Check {Min; Min+1 ;Max-1;Max }

Example of Boundary Value Analysis :

Requirement: Validate AGE field, which accepts values from 21-60.

We verify the following Boundary Value Test cases:

TC001: Validate AGE by entering 20 [Min-1]: Invalid Boundary Check

TC002: Validate AGE by entering 21 [Min]: Valid Boundary Check

TC003: Validate AGE by entering 22 [Min+1]: Valid Boundary Check

TC004: Validate AGE by entering 59 [Max-1]: Valid Boundary Check

TC005: Validate AGE by entering 60 [Max-1]: Valid Boundary Check

TC006: Validate AGE by entering 61[Max+1]: Invalid Boundary Check

The main advantage of Boundary Value Analysis is that the testing time is less as the tester will analyze the data only at the boundaries. Since this technique cannot concentrate on the errors that exist in the center of the input domain, it is always advisable to use BVA with a combination of Equivalence Class Partitioning.

EQUIVALENCE CLASS PARTITIONING (ECP)

- Equivalence Class Partitioning, the input domain values are partitioned into equivalent classes.
- It validates at least one value from each partition.
- So, the test cases are designed to cover at least one set of data from each equivalent class.
- Thus, it reduces the number of test cases.

- In other words, ECP concentrates on the possible errors at the center of the input domain instead of the boundaries.

Example: An online pharmacy website has the following offers for its customers.

If the cost of the medicines is 100 – 250, then a 0% discount.

If the cost of the medicines is 260 – 500, then a 5% discount.

10% discount if the cost is 510 – 1000.

A 15% discount, if the price is more than 1000.

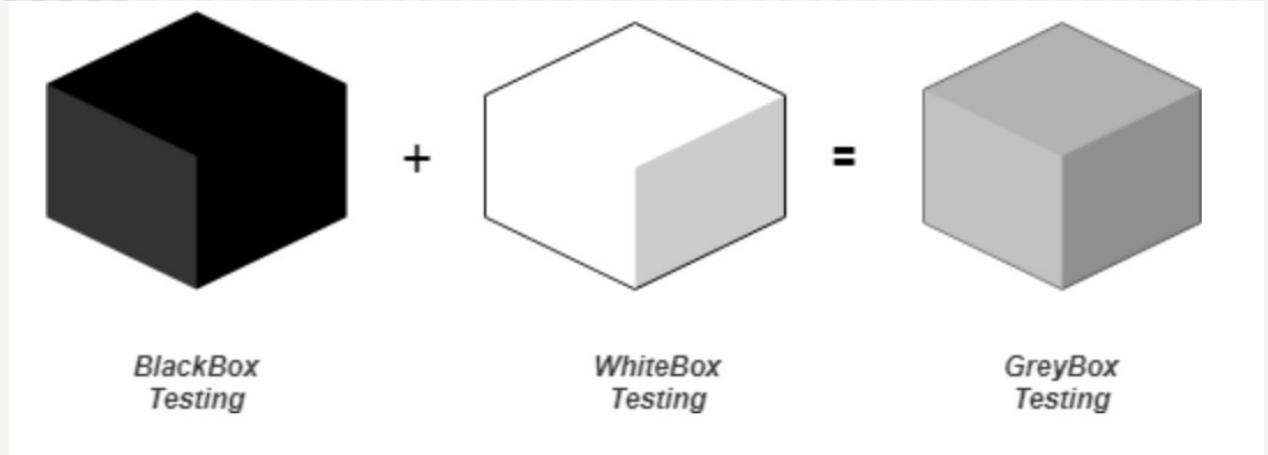
Here, the Equivalence Classes divides the following into valid/invalid cases.

Equivalence Classes	Input Value	Valid/Invalid
x < 100	50	INVALID
100 <= x <= 250	200	VALID
260 <= x <= 500	350	VALID
510 <= x <= 1000	780	VALID
x > 1000	150	VALID

GreyBox Testing

- Greybox testing is a software testing method to test the software application with partial knowledge of the internal working structure.

- It is a combination of black box and white box testing because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.



- GreyBox testing commonly identifies context-specific errors that belong to web systems.

- For example: while testing if tester encounters any defect then he makes changes in code to resolve the defect and then test it again in real time. It concentrates on all the layers of any complex software system to increase testing coverage. It gives the ability to test both presentation layer as well as internal coding structure. It is primarily used in integration testing and penetration testing.

- Grey Box Testing or Gray box testing is a software testing technique to test a software product or application with partial knowledge of internal structure of the application.

- The purpose of grey box testing is to search and identify the defects due to improper code structure or improper use of applications.

Example of Gray Box Testing: While testing websites feature like links or orphan links, if tester encounters any problem with these links, then he can make the changes straightaway in HTML code and can check in real time.

- **Matrix Testing:** This testing technique involves defining all the variables that exist in their programs.

- **Regression Testing**: To check whether the change in the previous version has regressed other aspects of the program in the new version. It will be done by testing strategies like retest all, retest risky use cases, retest within a firewall.

- **Orthogonal Array Testing or OAT**: It provides maximum code coverage with minimum test cases.

- **Pattern Testing:** This testing is performed on the historical data of the previous system defects. Unlike black box testing, gray box testing digs within the code and determines why the failure happened

Example

A simple black box testing example for a login functionality of a web application. In this scenario, we will test the login page without having access to the internal code or implementation details.

Test Case Name: Verify successful login with valid credentials.
Test Steps:

- Open the web browser.
- Enter the URL of the application's login page.
- Enter a valid username in the username field.
- Enter a valid password in the password field.
- Click on the "Login" button.
- Wait for the application to process the login request.

Expected Result: The user should be successfully logged into the application's dashboard/homepage.
Test Case Status: PASS (if the user is redirected to the dashboard/homepage)

Example

Test Case Name: Verify unsuccessful login with invalid credentials.
Test Steps:

- Open the web browser.
- Enter the URL of the application's login page.
- Enter an invalid username (e.g., "invaliduser") in the username field.
- Enter an invalid password (e.g., "wrongpassword") in the password field.
- Click on the "Login" button.

Wait for the application to process the login request.

Expected Result: The login attempt should fail, and an appropriate error message (e.g., "Invalid username or password") should be displayed on the login page.
Test Case Status: PASS (if the error message is displayed)

Features of Black Box Testing

- **Focus on External Behavior**: Black box testing emphasises evaluating the software's functionality from an end-user perspective, focusing on how the system behaves with different inputs and usage scenarios.
- **Independence from Internal Code**: Testers conducting black box testing do not require knowledge of the internal code or implementation details, making it suitable for testers who may not have programming expertise.
- **Requirement-Based Testing**: Test cases in black box testing are designed based on the software's requirements and specifications. This ensures that the application meets the intended functionality and business objectives.
- **Real-World Scenario Testing**: Black box testing helps simulate real-user conditions, allowing testers to identify defects that might arise during actual usage of the software.
- **Validation of Interfaces**: It is effective in verifying the accuracy of the software's interfaces, ensuring that inputs and outputs are correctly handled.

- **Identification of Interface-Level Bugs**: Black box testing is particularly useful for detecting interface-level bugs, such as incorrect error messages, incorrect data handling, or missing functionality.

- **User-Centric Testing**: By focusing on the end-user perspective, black box testing ensures that the application meets user expectations and delivers a satisfactory user experience.

- **Test Case Design Techniques**: Black box testing employs various test case design techniques, such as equivalence partitioning, boundary value analysis, decision table testing, and state transition testing, to ensure comprehensive test coverage.

- **Compatibility Testing**: It helps assess the software's compatibility with different environments, browsers, operating systems, and devices.

- Test Automation Support: Many black box testing tools support test automation, enabling the execution of repetitive test cases efficiently and reducing the testing cycle time.

Advantages of Blackbox testing

- **Independence from Internal Implementation:** Testers do not need to have access to the source code or knowledge of the internal implementation, making it suitable for non-technical team members.
- **User-Centric Testing:** Black box testing focuses on the software's external behavior, ensuring that it meets user requirements and expectations.
- **Testing from End-User Perspective:** It simulates real user scenarios, helping to identify usability issues and ensuring the software meets user needs.
- **Early Detection of Interface Issues:** Black box testing can uncover interface-related defects, such as input validation errors and output discrepancies.

- **Effective at Integration Testing:** It verifies the interactions between different system components, making it valuable for integration testing.

- **Test Case Design Flexibility:** Various test case design techniques, such as equivalence partitioning and boundary value analysis, allow for effective test coverage.

- **Effective for Requirement Validation:** Black box testing helps validate that the software meets the specified requirements.

- **Suitable for Large Projects:** It can be applied at different testing levels, from unit testing to acceptance testing, making it scalable for large projects.

Limitations of Black box testing

- **Limited Code Coverage:** Black box testing may not explore all possible code paths or internal logic, potentially leaving certain defects undetected.
- **Inability to Test Complex Algorithms:** It may not be effective at validating complex algorithms or intricate business logic that requires knowledge of the internal code.
- **Redundant Testing:** Some test cases may overlap, leading to redundant testing efforts and less optimal test coverage.
- **Dependency on Requirements:** Test cases are heavily dependent on the accuracy and completeness of the provided requirements. Incomplete or ambiguous requirements can result in incomplete testing.
- **Inefficiency with Repetitive Tasks:** Manual black box testing can be time-consuming and inefficient for repetitive tasks, making test automation essential for large-scale projects.
- **Inability to Assess Performance and Scalability:** Performance-related issues and scalability problems may not be effectively identified through black box testing alone.

- **Black-Box Testing is like a test drive**. You don't need to know how the engine works, how the brakes are connected, or how the transmission shifts gears. You're only interested in whether the car starts when you turn the key, moves when you press the accelerator, stops when you hit the brakes, and if the air conditioning and radio work. You're testing the functionality without knowing the internal workings.

- **White-Box Testing is like the inspection done by a mechanic**. The mechanic lifts the car, checks the engine, examines the brake system, looks at the transmission, etc. They deeply understand the car's workings and use this knowledge to check the internal components. They're interested in more than just whether the car works, but how it works.

QA testers

Black box - we do not know anything

Developers

White box - we know everything