Kalinga University

Faculty of Computer Science & IT

Course- BCAAIML
Subject- Software Engineering and Testing
Subject Code – BCAAIML503 **Sem- 5th**

Unit III

Principles of Software Testing

1. Testing shows the Presence of Defects

The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn't talk about the absence of defects. Software testing can ensure that defects are present but it cannot prove that software is defect-free. Even multiple tests can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects.

2. Exhaustive Testing is not possible

It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.

3. Early Testing

To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

4. Defect Clustering

In a project, a small number of modules can contain most of the defects. The Pareto Principle for software testing states that 80% of software defects come from 20% of modules.

5. Pesticide Paradox

Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

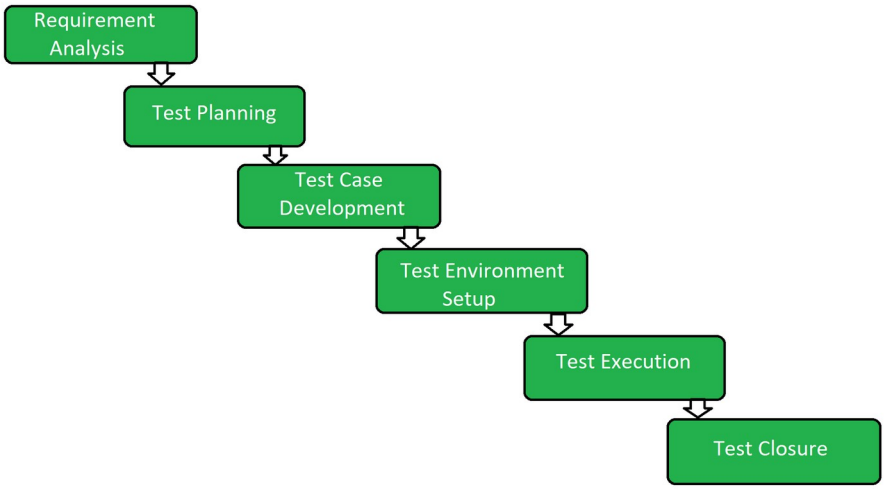
6. Testing is Context-Dependent

The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.

7. Absence of Errors Fallacy

If a built software is 99% bug-free but does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

Software Testing Life Cycle (STLC)



1. Requirement Analysis

Requirement Analysis is the first step of the **Software Testing Life Cycle (STLC)**. In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then the quality assurance team meets with the stakeholders to better understand the detailed knowledge of requirements.

The activities that take place during the Requirement Analysis stage include:

- Reviewing the software requirements document (SRD) and other related documents
- Interviewing stakeholders to gather additional information
- Identifying any ambiguities or inconsistencies in the requirements
- Identifying any missing or incomplete requirements
- Identifying any potential risks or issues that may impact the testing process

Creating a requirement traceability matrix (RTM) to map requirements to test cases

At the end of this stage, the testing team should have a clear understanding of the software requirements and should have identified any potential issues that may impact the testing process. This will help to ensure that the testing process is focused on the most important areas of the software and that the testing team is able to deliver high-quality results.

2. Test Planning

Test Planning is the most efficient phase of the software testing life cycle where all testing plans are defined. In this phase manager of the testing, team calculates the estimated effort and cost for the testing work. This phase gets started once the requirement-gathering phase is completed.

The activities that take place during the Test Planning stage include:

- Identifying the testing objectives and scope
- Developing a test strategy: selecting the testing methods and techniques that will be used
- Identifying the testing environment and resources needed
- Identifying the test cases that will be executed and the test data that will be used
- Estimating the time and cost required for testing
- Identifying the test deliverables and milestones
- Assigning roles and responsibilities to the testing team
- Reviewing and approving the test plan

At the end of this stage, the testing team should have a detailed plan for the testing activities that will be performed, and a clear understanding of the testing objectives, scope, and deliverables. This will help to ensure that the testing process is well-organized and that the testing team is able to deliver high-quality results.

3. Test Case Development

The **Test Case Development** phase gets started once the test planning phase is completed. In this phase testing team notes down the detailed test cases. The testing team also prepares the required

test data for the testing. When the test cases are prepared then they are reviewed by the quality assurance team.

The activities that take place during the Test Case Development stage include:

- Identifying the test cases that will be developed
- Writing test cases that are clear, concise, and easy to understand
- Creating test data and test scenarios that will be used in the test cases
- Identifying the expected results for each test case
- Reviewing and validating the test cases
- Updating the requirement traceability matrix (RTM) to map requirements to test cases

At the end of this stage, the testing team should have a set of comprehensive and accurate test cases that provide adequate coverage of the software or application. This will help to ensure that the testing process is thorough and that any potential issues are identified and addressed before the software is released.

4. Test Environment Setup

Test Environment Setup is an important part of the STLC. Basically, the test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process, the testing team is not involved. either the developer or the customer creates the testing environment.

5. Test Execution

In **Test Execution**, after the test case development and test environment setup test execution phase gets started. In this phase testing team starts executing test cases based on prepared test cases in the earlier step.

The activities that take place during the test execution stage of the Software Testing Life Cycle (STLC) include:

- **Test execution:** The test cases and scripts created in the test design stage are run against the software application to identify any defects or issues.
- **Defect logging:** Any defects or issues that are found during test execution are logged in a defect tracking system, along with details such as the severity, priority, and description of the issue.

- **Test data preparation:** Test data is prepared and loaded into the system for test execution
- **Test environment setup:** The necessary hardware, software, and network configurations are set up for test execution
- **Test execution:** The test cases and scripts are run, and the results are collected and analyzed.
- **Test result analysis:** The results of the test execution are analyzed to determine the software's performance and identify any defects or issues.
- **Defect retesting:** Any defects that are identified during test execution are retested to ensure that they have been fixed correctly.
- **Test Reporting:** Test results are documented and reported to the relevant stakeholders.

It is important to note that test execution is an iterative process and may need to be repeated multiple times until all identified defects are fixed and the software is deemed fit for release.

6. Test Closure

Test Closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented. The main objective of the test closure stage is to ensure that all testing-related activities have been completed and that the software is ready for release.

At the end of the test closure stage, the testing team should have a clear understanding of the software's quality and reliability, and any defects or issues that were identified during testing should have been resolved. The test closure stage also includes documenting the testing process and any lessons learned so that they can be used to improve future testing processes

Test closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented. The main activities that take place during the test closure stage include:

- **Test summary report:** A report is created that summarizes the overall testing process, including the number of test cases executed, the number of defects found, and the overall pass/fail rate.
- **Defect tracking:** All defects that were identified during testing are tracked and managed until they are resolved.

- **Test environment clean-up:** The test environment is cleaned up, and all test data and test artifacts are archived.
- **Test closure report:** A report is created that documents all the testing-related activities that took place, including the testing objectives, scope, schedule, and resources used.
- **Knowledge transfer:** Knowledge about the software and testing process is shared with the rest of the team and any stakeholders who may need to maintain or support the software in the future.
- **Feedback and improvements:** Feedback from the testing process is collected and used to improve future testing processes

It is important to note that test closure is not just about documenting the testing process, but also about ensuring that all relevant information is shared and any lessons learned are captured for future reference. The goal of test closure is to ensure that the software is ready for release and that the testing process has been conducted in an organized and efficient manner.

Quality Assurance (QA) vs Quality Control (QC)

Quality Assurance (QA) and Quality control (QC) are both important methods in software engineering to get high-quality software. Quality Assurance (QA) prevents software defects or minimizes the number of defects in software before delivery by making sure that proper methods and processes are followed during the software development process. Whereas Quality Control (QC) identifies and fixes the defects or errors that exist after development.

Quality Assurance (QA)

Quality assurance is a method of making the software application with fewer defects and mistakes when it is finally released to the end users. Quality Assurance is defined as an activity that ensures the approaches, techniques, methods, and processes designed for the projects are implemented correctly. It recognizes defects in the process. Quality Assurance is completed before Quality Control.

- It focuses on preventing defects.
- It is a proactive process and is preventive in nature.
- It helps to recognize flaws in the process.
- These activities monitor and verify that the processes used to manage and create deliverables have been followed.

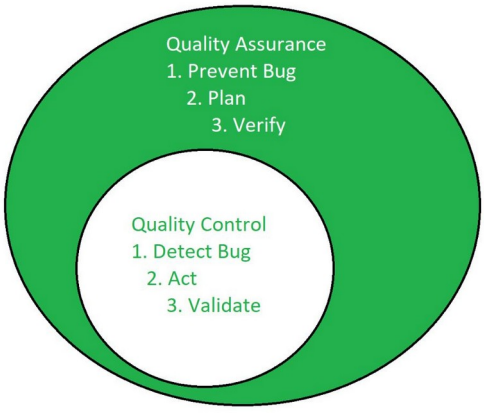
Characteristics of Quality Assurance

- **Process-Oriented:** To guarantee constant product quality, it places a strong emphasis on the creation and application of reliable procedures and standards.
- **Proactive Process:** Quality Assurance (QA) tries to prevent errors by enhancing procedures, guaranteeing that quality is integrated into the product from the start.
- **Continuous Improvement**: To improve quality over time, QA includes constant process assessment and improvement.

Quality Control

Quality Control is a software engineering process that is used to ensure that the approaches, techniques, methods, and processes designed for the project are followed correctly. Quality control activities operate and verify that the application meet the defined quality standards.

- It focuses on an examination of the quality of the end products and the final outcome rather than focusing on the processes used to create a product.
- It is a reactive process and is detection in nature.
- These activities monitor and verify that the project deliverables meet the defined quality standards.



Characteristics of Quality Control

- **Reactive Process:** It is the process to find and fix the flaws in the completed product, with focus on finding problems rather than solving them.

- **Defect Identification:** With testing and inspection, the goal is to find the defect as early as possible.
- **Product-oriented:** It entails examining, evaluating, and verifying the product to make sure it satisfies the required standards of quality.

Difference between Quality Assurance (QA) and Quality Control (QC)

Parameters	Quality Assurance (QA)	Quality Control (QC)
Objective	QA focuses on providing assurance that the quality requested will be achieved.	QC focuses on fulfilling the quality requested.
Technique	QA is the technique of managing quality.	QC is the technique to verify quality.
Involved in which phase?	QA is involved during the development phase.	QC is not included during the development phase.
Program execution is included?	QA does not include the execution of the program.	QC always includes the execution of the program.
Type of tool	QA is a managerial tool.	QC is a corrective tool.
Process/ Product-oriented	QA is process oriented.	QC is product oriented.
Aim	The aim of quality assurance is to prevent defects.	The aim of quality control is to identify and improve the defects.
Order of execution	Quality Assurance is performed before Quality Control.	Quality Control is performed after the Quality Assurance activity is done.
Technique type	QA is a preventive technique.	QC is a corrective technique.
Measure type	QA is a proactive measure.	QC is a reactive measure.
SDLC/ STLC?	QA is responsible for the	QC is responsible for

Parameters	Quality Assurance (QA)	Quality Control (QC)
	entire software development life cycle.	the software testing life cycle.
Activity level	QA is a low-level activity that identifies an error and mistakes that QC cannot.	QC is a high-level activity that identifies an error that QA cannot.
Focus	Pays main focus is on the intermediate process.	Its primary focus is on final products.
Team	All team members of the project are involved.	Generally, the testing team of the project is involved.
Time consumption	QA is a less time-consuming activity.	QC is a more time-consuming activity.
Which statistical technique was applied?	Statistical Process Control (SPC) statistical technique is applied to Quality Assurance.	Statistical Quality Control (SQC) statistical technique is applied to Quality Control.
Example	Verification	Validation

Verification and Validation in Software Engineering

Verification and Validation are the processes of investigating whether a software system satisfies specifications and standards and fulfills the required purpose. Verification and Validation both play an important role in developing good software. Verification helps in examining whether the product is built right according to requirements, while validation helps in examining whether the right product is built to meet user needs.

Verification

Verification is the process of checking that software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing. Verification means **Are we building the product right?**

Static Testing

Verification Testing is known as Static Testing, and it can be simply termed as checking whether we are developing the right product or not, and also whether our software is fulfilling the customer's requirement or not. Here are some of the activities that are involved in verification.

- **Inspections**: This is when we carefully look over the software's design or code to spot any potential problems or areas that could be improved.
- **Reviews**: This is a team effort, where everyone gets together to assess the software's requirements, design, or code. The goal is to make sure everything is on track and meets the specified objectives.
- **Walkthroughs**: In a walkthrough, the developer or designer gives a casual presentation of their work, explaining how it meets the project's requirements. It's a great way to get everyone on the same page.
- **Desk-Checking**: This is when a developer manually reviews their own code or design to catch any mistakes or issues before moving forward.

Validation

Validation is the process of checking whether the **Software Product** is up to the mark, or in other words product has high-level requirements. It is the process of checking the validation of the product i.e., it checks whether what we are developing is the right product. It is a validation of the actual and expected products. Validation is dynamic testing. Validation means **Are we building the right product?**

Dynamic Testing

Validation Testing is known as **Dynamic Testing** in which we examine whether we have developed the product right or not and also about the business needs of the client. Here are some of the activities that are involved in Validation.

- **Black Box Testing**: This is all about testing the software from the user's perspective. You don't need to know how the code works internally. Instead, you just focus on whether the software behaves as expected when you use it, like clicking buttons or entering information.
- **White Box Testing**: In this case, you have access to the internal workings of the software. You're testing the logic, the flow, and how the code is structured to make sure everything is functioning as it should behind the scenes.

- **Unit Testing**: This involves testing individual parts or functions of the software to make sure each piece works properly on its own, before they're combined with other parts of the system.
- **Integration Testing**: Once individual pieces are tested, integration testing checks how well different parts of the software work together. It's like making sure all the components play nicely when combined.

Differences between Verification and Validation

	Verification	Validation
Definition	Verification refers to the set of activities that ensure software correctly implements the specific function	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements.
Focus	It includes checking documents, designs, codes, and programs.	It includes testing and validating the actual product.
Type of Testing	Verification is the **Static testing**.	Validation is **Dynamic testing**.
Execution	It does ***not*** include the execution of the code.	It includes the execution of the code.
Methods Used	Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are **Black Box Testing**, **White Box Testing** and **Non-Functional testing**.

Black Box Testing

Black-Box Testing is a **Type of Software Testing** in which the tester is not concerned with the software's internal knowledge or implementation details but rather focuses on validating the functionality based on the provided specifications or requirements.

The following are the several categories of black box testing:

1. Functional Testing
2. Regression Testing
3. Nonfunctional Testing (NFT)

White Box Testing

White-box Testing is a Software Testing Technique that involves testing the internal structure and workings of a **Software Application**. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.
It is also called **Glass Box Testing** or **Clear Box Testing** or **Structural Testing**.

White box testing can be done for different purposes. The three main types are:

1. **Unit Testing**
2. **Integration Testing**
3. **Regression Testing**

Black Box Testing Vs White Box Testing

Parameters	**Black Box Testing**	**White Box Testing**
Definition	Black Box Testing is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	White Box Testing is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
Testing objectives	Black box testing is mainly focused on testing the functionality of the software, ensuring that it meets the requirements and specifications.	White box testing is mainly focused on ensuring that the internal code of the software is correct and efficient.
Testing methods	Black box testing uses methods like **Equivalence Partitioning**, **Boundary Value Analysis**, and **Error Guessing** to create test cases.	White box testing uses methods like **Control Flow Testing**, **Data Flow Testing** and **Statement Coverage Testing**.
Knowledge level	Black box testing does not require any knowledge of the internal workings of	White box testing requires knowledge of programming

Parameters	Black Box Testing	White Box Testing
	the software, and can be performed by testers who are not familiar with programming languages.	languages, software architecture and design patterns.
Scope	Black box testing is generally used for testing the software at the functional level.	White box testing is used for testing the software at the unit level, integration level and system level.
Implementation	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
Done By	Black Box Testing is mostly done by software testers.	White Box Testing is mostly done by software developers.
Terminology	Black Box Testing can be referred to as outer or external software testing.	White Box Testing is the inner or the internal software testing.
Testing Level	Black Box Testing is a functional test of the software.	White Box Testing is a structural test of the software.
Testing Initiation	Black Box testing can be initiated based on the requirement specifications document.	White Box testing of software is started after a detail design document.
Programming	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
Testing Focus	Black Box Testing is the behavior testing of the software.	White Box Testing is the logic testing of the software.
Applicability	Black Box Testing is applicable to the higher levels of testing of software.	White Box Testing is generally applicable to the lower levels of software testing.
Alternative Names	Black Box Testing is also called closed testing.	White Box Testing is also called as clear box testing.

Parameters	Black Box Testing	White Box Testing
Time Consumption	Black Box Testing is least time consuming.	White Box Testing is most time consuming.
Suitable for Algorithm Testing	Black Box Testing is not suitable or preferred for algorithm testing.	White Box Testing is suitable for algorithm testing.
Approach	Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example	Search something on google by using keywords	By input to check and verify loops
Exhaustiveness	It is less exhaustive as compared to white box testing.	It is comparatively more exhaustive than black box testing.

Static Testing Techniques

As explained earlier, Static Testing techniques are testing techniques that do not require the execution of a code base.

Static Testing Techniques are divided into two major categories:

1. **Reviews:** They can range from purely informal peer reviews between two developers/testers on the artifacts (code/test cases/test data) to formal **Inspections** which are led by moderators who can be internal/external to the organization.

1. **Peer Reviews:** Informal reviews are generally conducted without any formal setup. It is between peers. For Example- Two developers/Testers review each other's artifacts like code/test cases.
2. **Walkthroughs:** Walkthrough is a category where the author of work (code or test case or document under review) walks through what he/she has done and the logic behind it to the stakeholders to achieve a common understanding or for the intent of feedback.
3. **Technical review_:** It is a review meeting that focuses solely on the technical aspects of the document under review to achieve a consensus. It has less or no focus on the identification of defects based on reference documentation. Technical experts like architects/chief designers are required to do the review. It can vary from Informal to fully formal.

4. **Inspection:** Inspection is the most formal category of reviews. Before the inspection, the document under review is thoroughly prepared before going for an inspection. Defects that are identified in the Inspection meeting are logged in the defect management tool and followed up until closure. The discussion on defects is avoided and a separate discussion phase is used for discussions, which makes Inspections a very effective form of review.

2. **Static Analysis:** Static Analysis is an examination of requirement/code or design to identify defects that may or may not cause failures. For Example- Review the code for the following standards. Not following a standard is a defect that may or may not cause a failure. Many tools for Static Analysis are mainly used by developers before or during Component or Integration Testing. Even **Compiler** is a Static Analysis tool as it points out incorrect usage of syntax, and it does not execute the code per se. There are several aspects to the code structure - Namely Data flow, Control flow, and Data Structure.

1. **Data Flow:** It means how the data trail is followed in a given program - How data gets accessed and modified as per the instructions in the program. By Data flow analysis, you can identify defects like a variable definition that never got used.
2. **Control flow:** It is the structure of how program instructions get executed i.e. conditions, iterations, or loops. Control flow analysis helps to identify defects such as Dead code i.e. a code that never gets used under any condition.
3. **Data Structure:** It refers to the organization of data irrespective of code. The complexity of data structures adds to the complexity of code. Thus, it provides information on how to test the control flow and data flow in a given code.

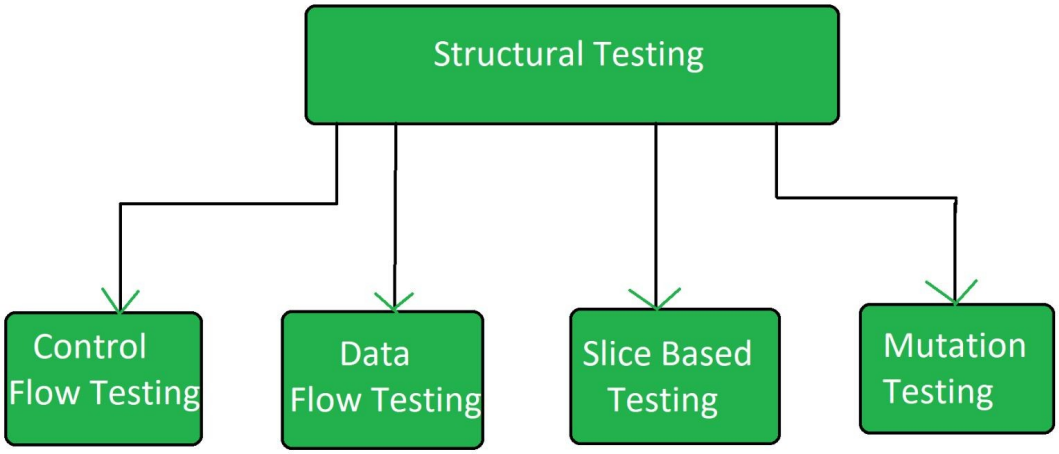
Structural Testing

Structural testing is a type of software testing that uses the internal design of the software for testing or in other words the software testing which is performed by the team which knows the development phase of the software, is known as structural testing.

Structural testing is related to the internal design and implementation of the software i.e. it involves the development team members in the testing team. It tests different aspects of the software according to its types. Structural testing is just the opposite of behavioral testing.

Types of Structural Testing

There are 4 types of Structural Testing:



Control Flow Testing:

Control flow testing is a type of structural testing that uses the programs's control flow as a model. The entire code, design and structure of the software have to be known for this type of testing. Often this type of testing is used by the developers to test their own code and implementation. This method is used to test the logic of the code so that required result can be obtained.

Data Flow Testing:

It uses the control flow graph to explore the unreasonable things that can happen to data. The detection of data flow anomalies are based on the associations between values and variables. Without being initialized usage of variables. Initialized variables are not used once.

Slice Based Testing:

It was originally proposed by Weiser and Gallagher for the software maintenance. It is useful for software debugging, software maintenance, program understanding and quantification of functional cohesion. It divides the program into different slices and tests that slice which can majorly affect the entire software.

Mutation Testing:

Mutation Testing is a type of Software Testing that is performed to design new software tests and also evaluate the quality of already existing software tests. Mutation testing is related to

modification a program in small ways. It focuses to help the tester develop effective tests or locate weaknesses in the test data used for the program.

Advantages of Structural Testing

- It provides thorough testing of the software.
- It helps in finding out defects at an early stage.
- It helps in elimination of dead code.
- It is not time consuming as it is mostly automated.

Disadvantages of Structural Testing

- It requires knowledge of the code to perform test.
- It requires training in the tool used for testing.
- Sometimes it is expensive.

White Box Testing Challenges

1. **High Technical Knowledge Requirement**
   Testers must deeply understand code logic, architecture, and programming languages.

2. **Incomplete Code Coverage**
   Achieving 100% coverage (statements, branches, paths) is difficult and time-consuming.

3. **Time-Consuming for Large Codebases**
   Manually writing and maintaining unit-level test cases is resource-intensive.

4. **Maintenance Overhead**
   Tests may break with frequent code changes, requiring constant updates.

5. **Limited Focus on UI/UX**
   Since it's logic-focused, it often misses usability or interface-related issues.

6. **Inadequate for Missing Functionality**
   It assumes all requirements are implemented, so missing features might go undetected.

7. **Not Effective for High-Level Testing**
   Integration and system-level behaviors are hard to evaluate.

8. **Difficult with Legacy or Poorly Documented Code**
   Analyzing and testing old or spaghetti code is often impractical.

9. **Security Holes May Be Missed**
   Unless security testing is specifically included, vulnerabilities might be overlooked.

10. **Tool Limitations**
    Not all tools support complex logic or dynamic constructs like reflection or runtime code generation.

Black Box Testing Challenges

1. **Incomplete Test Coverage**
   Without internal code knowledge, some paths may remain untested.

2. **Ambiguous Requirements**
   Lack of clear documentation leads to incorrect or missing test cases.

3. **Redundancy in Test Cases**
   Testers may unknowingly test the same logic multiple times.

4. **Limited Input Combination Testing**
   Testing all possible input combinations is infeasible for complex systems.

5. **Not Suitable for Performance or Load Testing Alone**
   Needs integration with other testing types for complete coverage.

6. **Difficult to Trace Bugs**
   Debugging is harder as testers don't know the internal code where failures occur.

7. **Time-Consuming for GUI Testing**
   Manual GUI testing is slow and often error-prone.

8. **Missing Logical Errors**
   Some logic errors might only be detectable with knowledge of the code.

9. **Poor Tool Support for Complex Systems**
   Automated tools may struggle to simulate realistic usage scenarios.

10. **Test Case Maintenance**
    UI changes, even minor ones, can break automated black-box tests.