BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 6

DCA3245

SOFTWARE PROJECT MANAGEMENT

Unit 12

Testing Techniques

Table of Contents

SL No	Topic	Fig No / Table / Graph	SAQ / Activity	Page No
1	Introduction	-	-	3
	1.1 Objectives	-	-	
2	Software Testing Concepts	-	-	4 - 9
3	Types of Software Testing	-	-	10 - 18
	3.1 Manual testing	-	1	
	3.2 Automated testing	-	-	
4	Black Box Testing	-	2	19 - 22
5	White Box Testing Techniques	1, 2, 3	3	23 - 33
6	Summary	-	-	34
7	Terminal Questions	-	-	35
8	Answers	-	-	36

1. INTRODUCTION

In the previous unit, we have discussed various automated tools available for developing software. Once the software is developed, the next step is to test it so that it satisfies customer needs. In this unit, we shall discuss various software testing techniques.

1.1 Objectives:

After studying this unit, you should be able to:

- Explain various software testing concepts
- Describe types of software testing
- Discuss black box and white box testing
- List out various testing tools

2. SOFTWARE TESTING CONCEPTS

Software testing is the process of running through the application or a software product with the intention of uncovering the errors. It is the mechanism by which one can ascertain that the product meets its intended specifications. This is different from other physical process which has a predefined set of inputs and outputs. Because of this predefined input and output, the physical systems fail in a particular way. Same cannot be said for the software systems; software can fail in multiple types because of the random nature of the behavior. Hence software testing is also aimed at determining the different modes of failure as much as possible.

In the physical system, the main source of defect is the manufacturing defect. Also the physical products suffer from the factors like wear and tear, corrosion etc. The software product on the other hand does not have these factors; instead the main source will be the design defects. Further as a correlation to physical products, the software products might suffer from poor coding, but the major defects are the design flaws. These design bugs remain hidden under the product shipped and surface when a particular scenario occurs in the production environment needing immediate solution for the problem. Also, it is impossible to make any software product defect free; there will be some or other defects as long as you keep on testing the product. The team defines minimum exit criteria, and if the set of requirements and the exit criteria are met the software product will be released to production.

As stated earlier, eliminating all the defects from the software product is nearly impossible. This is because of the complex nature of the software and also the dynamic behavior. The software system is not a continuous system; hence finding the boundary conditions is difficult. Also, the software input can be very unpredictable. Covering all these scenarios or possible inputs to the software during the software testing phase is a very difficult task. On the similar lines, we might not be able to provide all the possible inputs and test the expected output of the application. For example, consider a software program that performs addition of two 8 bit numbers. In order to completely test the software, you have key in all the possible 256 combinations of inputs for which team will not have neither enough time or budget. Also, the small amount of code changes might have a huge impact on the behavior of the system

and might introduce some other defects in the system. For this purpose every code change might require regression testing of entire software application which is both a time consuming activity and results in increased cost to the organization. Hence, team might not be encouraged to take up such fixes that are not considered significant in project's context. Because of all these we might expect some amount of defects left uncovered in the system.

Regardless of the limitations stated above, testing is a very important aspect of the software development. Approximately 50 percent of the development time is spent on testing the software. Testing is integral part of the software development, and every release of the application can be released only after it has gone through proper testing.

Testing is usually performed for the following purposes:

To improve quality

The foremost reason of testing is to improve the quality of the software product being delivered. Quality at the fundamental level means conformance to requirements. The product delivered must meet the requirements of the customer. The behavior of the product should be according to the functional specifications defined for the product. Testing ensures that the application is run through different possible scenario's to confirm the behavior is as per specs.

Any deviation in behavior of the product from the requirements is termed as a defect or bug in the application. The defects can be very costly and result in some major mishaps if remain in the system. For example a defect in air control system might lead to air crash causing human life losses. Hence the prime goal of the testing is to uncover the defects in the system and help programmers to debug the application to make the application defect free and of highest quality.

For verification & validation (V&V)

Verification and Validation is a very important concept of the software development activity. Verification ensures that the product implementation is being carried out in the right manner whereas the validation ensures that right product is implemented. In other words,

the verification is like review activity like code review, design review etc. and validation corresponds to the testing of the product against the requirements.

The quality of a software product cannot be tested; however the result of testing works as the measurement criteria for the quality of the product being delivered. The quality is determined by three factors – the functionality, the engineering skills, and adaptability of the product. These three sets of factors are considered as dimensions in the software quality space. Each dimension may be further elaborated and its components can be further considered in greater details.

Software testing can be considered as a process and as well as a separate discipline. Even though the software testing is the part of the software development lifecycle, it has to be carried out as an independent activity. Software testing should have its own plan and should be tracked as independent project. In this way the software testing being logically an independent module of the project will be able to hold up or slow down the product delivery if the predefined quality goals are not met by the product development team. In that sense testing is a separate discipline from software development.

From the implementation perspective, the testing is an iterative process validating the functionality of the software and also the process attempting to break the software. The iterative process of software testing consists of:

- Designing test cases
- Executing tests
- Identifying defects
- Getting problems fixed and validating the fix

The goal of software testing is to find bugs and fix them to improve quality of the software. Software testing typically represents 40% of a software development budget.

Software Testing Techniques

There are two phases involved in software testing

1. Constructive Phase

2. Destructive Phase

Under constructive phase we have: Analysis, Design and Implementation Process. Constructive testing's major objective is to guarantee that the product delivers as promised and fulfils its stated purpose.

Under Destructive Phase we have Testing process: Destructive testing is performed with the objective of making the programme malfunction so that its weaknesses can be exposed.

A thorough examination of a software system requires both types of testing, constructive and destructive. While destructive testing can assist find and fix vulnerabilities, constructive testing can make sure the software can do what it's supposed to. Together, they help build more secure and trustworthy programmes.

Software Testing Principles:

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

- Testing shows the presence of defects: The application will undergo rigorous testing by the test engineer to guarantee the absence of any faults. During testing, we can only see if the programme or app has flaws. Because the entire test should be traceable to the customer requirement, which means to find any defects that might cause the product failure to meet the client's needs, the primary purpose of doing testing is to identify the number of unknown bugs with the help of various methods and testing techniques.
- Exhaustive Testing is not possible: The application will undergo rigorous testing by the test engineer to guarantee the absence of any faults. During testing, we can only see if the programme or app has flaws. Because the entire test should be traceable to the customer requirement, which means to find any defects that might cause the product

failure to meet the client's needs, the primary purpose of doing testing is to identify the number of unknown bugs with the help of various methods and testing techniques

- Early Testing: In this context, "early testing" refers to the practise of beginning the testing process as soon as possible during the software development life cycle's requirement analysis phase in order to catch any potential problems as soon as they can be fixed at a lower overall cost
- Defect Clustering: Defect clustering is a method for identifying groups of similar issues that might be found throughout testing. Many factors contribute to this, including the potential complexity of the modules and the coding phase.
- Pesticide Paradox: This concept states that repeat execution of the same collection of test cases over a period of time will not reveal any new faults in the programme being tested. Reviewing all of the test cases regularly is crucial for resolving the pesticide contradictions. And the implementation of various portions of the application or software necessitates the writing of new and varied tests, which in turn allows us to discover more defects.
- Testing is context-dependent: Various types of testing, including e-commerce, commercial, and so on, are available because of this context-dependence principle. Due to the fact that each application has its own set of requirements, features, and functionality, there is a clear method for testing both commercial and e-commerce websites.
- Absence of errors fallacy: After rigorous testing has been performed and all known issues have been resolved, the programme can be considered to be nearly bug-free. However, there is always a potential that testing the programme alongside the wrong specifications, finding the bugs, and fixing them within a certain time frame won't assist because testing is performed on the wrong specification, which doesn't correspond to the needs of the client.

Software Testability checklist:

Checklist - is a list of tests which should be run in a definite procedure. It helps to understand if testing is fully run and how many failed.

- Operability (the better it works the more efficiently it can be tested)

- Observability (what you see is what you test)
- Controllability (the more software can be controlled the more testing can be automated and optimized)
- Decomposability (by controlling the scope of testing, the more quickly problems can be isolated and retested intelligently)
- Simplicity (the less there is to test, the more quickly we can test)
- Stability (the fewer the changes, the fewer the disruptions to testing)
- Understandability (the more information known, the smarter the testing)

3. TYPES OF SOFTWARE TESTING

The Software testing consists of several sub categories, each subcategory created for separate purposes and different techniques. Commonly created Software testing categories include:

- *Functional testing* to verify that the functionality implemented is as per the specifications. It may include verification of business rules, validation of the formulae, calculation, and testing of user interfaces against the wire frames etc.
- *Forced error testing*- in forced error testing deliberate effort is made to introduce the error in the application flow trying to ensure that the system exists gracefully.
- *Compatibility testing* to ensure that software is compatible with various hardware platforms, operating systems, other software packages, and even previous versions of the same application.
- *Performance testing* to determine the response time of the application and compare it with the non-functional requirements of the product specification.
- *Scalability testing* to ensure that the software will function well when the number of users is increased and system behaves predictably when more number of servers and database are added to the production set up.
- *Stress testing* to see how the system performs under extreme conditions, such as a very large number of concurrent users using the system.
- *Usability testing* to ensure that the software is easy to use and operate.
- *Application security testing* to ensure that the customer data access happens only through known channels with proper certification and data is safe.

Depending on the type of the software product some additional testing also may be needed like compliance testing, accessibility testing etc. There are two basic methods of performing software testing:

1) Manual testing
2) Automated testing

3.1 Manual Testing

As the name suggests manual testing involves a tester or group of testing team members perform the testing activity manually. The testing might involve navigating through the user interfaces, performing a transaction like online order submission, or even trying some scenario that could break the system. Manual testing is time consuming as well as labor intensive work. However manual testing is required for some kind of testing opportunities, these include:

- User interface testing of a newly released software and areas of application that undergo lot of changes
- Usability testing
- Scenario based testing wherein the testing need not follow any particular scenario, rather would be based on some ad-hoc testing.
- UAT – user acceptance testing

As stated earlier, manual testing is a very time consuming activity. To fully test a system typically manual testing takes from weeks to months. Also, another drawback of the manual testing is the dependency on testing resources. As the result directly is based on the efficiency of the testing resources, the result cannot be standardized. Because of these reasons many organizations look to automate the routine rounds of testing as quickly as possible to save time and standardize the test results.

3.2 Automated Testing

Automated testing is the method of software testing with minimal or no manual intervention. In this technique, the test scripts are initially recorded once and the test scripts are run using the test runner tool. The input can remain constant or can be dynamically changed during the test execution. Another feature of the automated testing is that these tests can be run again and again. If the automation scripts are designed and recorded properly, the variance in test result which is the main drawback of manual testing can be eliminated.

Before deciding on automating software testing, enough diligence must be applied on the usefulness of adopting automated testing. It is better to avoid automated testing and perform the testing manually if the process of automating a test scenario calls for huge effort, if a part of the software undergoes lot of changes, or if the testing is done not more than 2-3 times.

As automated testing is a process of test script development and execution using the test runner, automation process must be carefully designed for the organization. The ultimate goal of using the test automation is to achieve the repetitive run of the testing in minimum time producing standard test results. The test automation should be well architected so that the test scripts minimum maintenance and help organization in attaining higher quality standards.

History of Testing:

Phase	Year	Period	Detail
I	Before 1956	The Debugging-Oriented Period	Testing to make sure that the software satisfies its specification
II	1957 ~78	The Demonstration-Oriented Period	Testing to detect implementation faults
III	1979~82	: The Destruction-Oriented Period	Testing to detect implementation faults
IV	1983~97	The Evaluation-Oriented Period	Testing to detect faults in requirements and design as well as in implementation
V	Since 1988	The Prevention-Oriented Period	Testing to prevent faults in requirements, design, and implementation

SELF-ASSESSMENT QUESTIONS -1

1. Software testing is the process of running through the application or a software product with the intention of uncovering the errors. (True / False)
2. _____ to ensure that the software will function well when the number of users is increased on the system.
3. Software testing typically represents _____ of a software development budget. (Pick right option)
   a) 40%
   b) 20%
   c) 70%
   d) 90%

Testing classification by purpose:

Different testing techniques are designed depending on the place and scope where these will be used in the software development life cycle. On the broader classification of testing its been grouped into four types of testing

- Correctness testing
- Performance testing
- Reliability testing
- Security testing

- Correctness testing is generally used in the field of software engineering, a program or system is considered proper when it operates in every one of its intended use cases. Engineers create a set of requirements that their system must meet in order to be correct before any software is written.
- The process of assessing a system's responsiveness and stability under a specific workload is called performance testing. Performance tests are usually run in order to assess application size, robustness, speed, and reliability.

- Software testing that assesses a system's capacity to operate as intended over a prolonged period of time without interruption is known as reliability testing. Reliability testing aims to find and fix problems that could lead to system failure or unavailability.
- Security testing is a subset of software testing that looks for system weaknesses and establishes if the system's resources and data are safe from potential hackers. It guarantees that there are no dangers or hazards to the software system or application that could result in a loss.

Testing classification by Life cycle Phases:

- Testing has been classified based on the lifecycle phases: so we have around six types of testing depending on the life cycle activities:
- Requirement phase testing: Test cases, conditions, and data in requirements-based testing are all derived directly from the requirements themselves.
- Design Phase testing : Product design testing is an integral aspect of the design thinking process that yields better answers to customer issues. After the team has identified user JTBDs and defined the difficulties they face, they will brainstorm potential solutions and create prototypes.
- Program Phase Testing : Verifying the expected behaviour of a programme through its actual execution. The execution of the programme, alongside the addition of test data, is followed by an analysis of the program's behaviour in response to the input. A comparison of the correctness proofs of two different programmes.
- Evaluating Phase Testing : The process of using a program's actual execution to confirm that it behaves as predicted. After the code has been run and some test data has been added, its response to the input is analysed. Two programmes' proofs of correctness are compared and contrasted.
- Installation Phase Testing : is a subset of software testing that double-checks the software's setup and makes sure it runs smoothly on a wide range of hardware and operating systems.
- Acceptance Phase Testing : is a method of quality assurance (QA) used to evaluate how well an app is received by its target audience. Beta testing, application testing, field testing, and end-user testing are all types of acceptance testing that can be performed.

Testing classification by scope:

Different testing techniques are designed depending on the place and scope where these will be used in the software development life cycle.

Here the testing is classified based on Scope

1. Unit testing
2. Component Testing
3. Integration Testing
4. System Testing

- **Program:** The art of testing consists of creating, selecting, exploring, and revising models.

  Our ability to go through this process depends on the number of different models we have at hand and their ability to express a program's behavior. Generally programs are very complicated to understand in detail. The program needs to be simplified in order to test it.

  If the model of the program is not able to explain the unexpected behaviour, model need to be modified to include more facts and details.

  Word processors, web browsers, database managers, and even scientific calculators are just a few examples of the many tasks that may be accomplished with the help of specialised software. Developers of software programmes use programming languages to create the software, which is subsequently compiled or translated into machine code. As a result of their contributions to modern computing, computers are useful in many industries, from commerce and science to entertainment and beyond.

  The goals and preferences of the intended viewers determine the program's specific design. Features are typically prioritised by developers according to user needs, market trends, and the software's expected function.

The program's model may need to be updated to account for more specifics if it cannot account for some form of unexpected behaviour.And if this fix doesn't work, the whole programme will have to be redone.

- **Bug:**

A "bug" is a defect or weakness in software that causes it to act in a way that was not intended or is otherwise inappropriate. These flaws might be very subtle, like a misspelt word in the user interface, or very severe, like a crash or data corruption. A program's code, design, documentation, or even its requirements might all have errors.

Optimistic notions about bugs: Benign Bug Hypothesis: The belief that bugs are nice, tame and logical.

Bugs are more deceptive than we expect them to be.

An unexpected test result may force us to change our idea about a bug and our model of bugs. Some optimistic notions that many programmers or testers have about bugs are usually unable to test effectively and unable to justify the dirty tests most programs need

These are errors in the code that cause the programme to fail to compile or run because they go against the rules of the programming language.

Faulty Reasoning: Errors in the algorithm or the logical flow of the code might lead to inaccurate results even if the syntax is fine.

Failures that occur during execution: These manifest themselves during execution and can cause the programme to crash or terminate unexpectedly. Common instances are division by zero and null pointer exceptions.

- **TEST:**

As used here, "test" refers to a defined series of steps taken under controlled settings to evaluate some aspect of a programme or system's operation.

The objective of testing is to find and fix bugs in the programme before releasing it to the public.Functional requirements, performance benchmarks, and other specifications are used to guide the development of tests for the software. The purpose of these tests is to ensure that the software operates as intended across a variety of use cases.Tests are formal techniques, Inputs must be prepared, Outcomes should predict, tests should be documented, commands need to be executed, and results are to be observed.

We do three distinct kinds of testing on a typical software system. They are:

- Unit / Component Testing
- Integration Testing
- System Testing

a. Exhaustive Testing:

- Exhaustive testing is a technique used in software development that tests all possible combination of data set . This sort of testing is used to ensure that the program does not crash under any circumstance.

  Exhaustive testing, also known as "complete testing" or "all-paths testing," is a theoretical concept in software testing that involves testing every possible input combination, scenario, and path through a program. In theory, this would ensure that every aspect of the software has been thoroughly tested, leaving no room for undetected defects or issues.

  However, in practice, achieving exhaustive testing is often impossible, especially for non-trivial software applications. This is due to the potentially astronomical number of combinations and scenarios, especially in complex systems.

  As used here, "test" refers to a defined series of steps taken under controlled settings to evaluate some aspect of a programme or system's operation. The objective of testing is to find and fix bugs in the programme before releasing it to the public. Functional requirements, performance benchmarks, and other specifications are used to guide the development of tests for the software. The purpose of these tests is to ensure that the software operates as intended across a variety of use cases.

b. Selective Testing:

- Selective Testing techniques are an approach of performing regression testing over a software product, using a minimal and specific set of test cases, available is test suites. These test cases can be selected through three different ways for selectively retesting software, which are: coverage Technique.
- Selective testing, also known as "focused testing" or "risk-based testing," is a strategy for software testing that places more emphasis on evaluating fewer, but more critical, features and/or components of an application. The idea is to pinpoint the most important parts of the product, the weak spots, and the places where bugs are most likely to appear, then focus testing efforts there.
- When time and money are at a premium during software testing, selective testing is an efficient and productive strategy. It enables testing teams to focus their efforts where they will have the biggest positive effect on the software's quality and dependability.

4. BLACK BOX TESTING

Black Box Testing attempts to derive sets of inputs that will fully exercise all the functional requirements of a system. Black box testing tries to determine the defects in the following classification:

1) Erroneous or unimplemented functionalities
2) User Interface errors
3) Errors in access of external interface like database connectivity
4) Performance issues of the application
5) Application initiation and termination errors.

Black box tests are intended to answer the below questions:

1) How can we ensure that the functionality is valid?
2) Which categories of input make good test cases?
3) Will the system behave abnormally for certain set of inputs?
4) What are the boundary values for data input?
5) What is the growth rate and volumes of data that system can tolerate?
6) What would be the effect of a particular combination of data on the system?

While the white box testing is carried out during the early stages of code development, the black box testing is carried out during the later stages on the integrated code. White box test cases are developed keeping the following targets in consideration

1) Cut down on the additional test cases to be developed in order to have a satisfactory coverage of functionality.
2) Test cases should concentrate on all the possible different errors that can be resulted from the system rather than concentrating on only one type of error.

Now let's see various methods of black box testing.

Equivalence partitioning

This method divides the input domain of a program into classes of data from which test cases can be derived. Equivalence partitioning aims to group all the test cases that generate same

type of error so that number of test cases needed to test the entire application can be reduced. It is based on an evaluation of equivalence classes for an input condition. An equivalence class represents a set of valid or invalid states for input conditions.

Equivalence classes may be defined according to the following guidelines:

1) If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
2) If an input condition requires a specific value, then one valid and two invalid equivalence classes are defined.
3) If an input condition specifies a member of a set, then one valid and one invalid equivalence classes are defined.
4) If an input condition is Boolean, then one valid and one invalid equivalence class are defined.

Boundary value analysis

This method leads to a selection of test cases that exercise boundary values. In a way the boundary value analysis complements the equivalence partitioning (EP) we just saw. In EP method the input selected is within the range of inputs, whereas in the BVA the inputs are selected on the edge of possible input spectrum of values. BVA focusses on output domain also. BVA guidelines include:

1) If the input values are bounded by x and y, the test cases should include the both the values x, and y, and also value just lower than x and higher than y.
2) Similarly if the inputs are specified by numerous values rather than specified in terms of range, then the test cases should include minimum and maximum values that are just below and above these values.
3) Above two guidelines should be applied to the output.
4) The boundary conditions should be determined for data structures used by the program and corresponding test cases should be developed.

Cause – effect graphing techniques

Cause-effect graphing technique provides logical representation of various possible operational scenarios of the application being tested. Here causes represent the input condition and effect is the system response. There are four steps:

1) Assign an identifier to each of the cause (input) and effect (output) applicable for a module.
2) Develop the cause-effect graph.
3) Develop the decision table based on the analysis of the graph.
4) Analyze the Decision table data and convert the same to test cases.

SELF-ASSESSMENT QUESTIONS -2

4. Black Box Testing tests the internal structure of the system. (True / False)
5. _____ method leads to a selection of test cases that exercise boundary values.
6. _____ technique provides logical representation of various possible operational scenarios of the application being tested. (Pick right option)
   a) Boundary Value Analysis
   b) Cause-effect graphing
   c) Equivalence Partitioning
   d) Basis Path Testing

Advantages of Black Box Testing:

- Effective for massive infrastructures. Testing is fair and unbiased when the tester and developer are not affiliated with one another in any way.
- Anyone can play the role of tester.The tester's lack of in-depth functional expertise of the system is irrelevant. Since the system's ultimate acceptance rests with its end users, testing will be conducted from their perspective.

- Inconsistencies and ambiguities in functional specifications might be revealed through testing.
- Once the functional specs are finalised, test cases can be developed. Of the test classes unique to black box testing, of special importance are system performance tests such as load tests and availability tests
- Black box testing offers to perform the majority of testing classes, majority of which can be implemented only by black box tests
- Black box testing needs less resources than white box testing for the software packages where the testing classes can be carried out by both black box and white box

Disadvantages of Black Box Testing:

- Without precise functional specifications, designing test cases is difficult.
- Without developing test cases based on specifications, it is difficult to recognise challenging inputs. Finding all possible inputs in a short amount of time might be challenging. This could make the process of creating test cases tedious and time-consuming.
- During testing, there is always a risk of discovering previously unknown directions. It's likely that the developer will have to run the same tests again.
- Possibility that coincidental aggregation of several errors will produce the correct response for a test case, and prevent error detection.
- Black box tests may not execute a substantial proportion of the code lines, which are not covered by a set of test cases.
- Black box tests is not able to easily identify cases of errors that counteract each other to accidentally produce the correct output.
- There is no control of line coverage. In situations where black box testers wish to improve line coverage, it is extremely difficult to specify the parameters of the test cases required to improve coverage.
- Consequently, testing the quality of coding and its strict adherence to the coding standards is impossible

5. WHITE BOX TESTING TECHNIQUES

- White box testing is a test case design method that uses the control structure of the procedural design to derive test cases. James Bach is well-known in the software testing community for his support of exploratory and context-based testing. Through his papers, talks, and courses, he has made major contributions to the testing community. James Bach has put a lot of emphasis on the idea of "testability." His definition of testability is how well and quickly a software system or its component can be tested. Several aspects that affect the efficiency and convenience of testing are included.
- "Testability is the measure of how easy or difficult it is to design, implement, execute, and interpret tests on a system. It is influenced by factors such as the clarity of requirements, the availability of testing tools, the level of automation, the accessibility of system components, and the degree to which the system supports various testing techniques."
- Testability refers to the degree to which a system may be tested effectively after its design and development. A highly testable system is one that can have its behaviours easily identified and verified by testers, allowing them to quickly and easily find bugs or other flaws. The idea behind this phrase is to keep testing in mind at every stage of the software creation process.

These attributes suggested by Bach can be used to develop software work products that are amenable to testing. Soft-ware testability is simply how easily a computer program can be tested

- The following characteristics lead to testable software
- Operability
- Observability
- Controllability
- Decomposability
- Simplicity
- Stability

- Understandability

1. Operability: The better it works, the more efficiently it can be tested." If the development team has quality in mind while designing and implementing a system, reasonably few bugs will be detected, thus allowing testing to progress without any hinderance

   The term "operability" is used to describe how simple and straightforward it is for users or administrators to manage a given system or application. It includes all the activities necessary to keep a system running smoothly and efficiently, including management, monitoring, and support. Designing and managing systems so that they are easily operable is essential. Highly operable systems are less complicated to set up, control, and maintain, which boosts their dependability and efficiency.

2. Observability: Observability: With the use of outputs, logs, metrics, and other observable data, software and systems engineers can acquire insights into a system's inner workings, behaviour, and performance. In today's highly dynamic contexts, such as microservices architectures and cloud-based applications, observability is more important than ever. Insights into system behaviour, problem diagnosis, and educated judgements about performance optimisation, troubleshooting, and system improvement are all made possible by this data.

   System states and variables are visible or verifiable during execution.

   - Source code is accessible.
   - "What you see is what you test." Inputs provided as part of testing generate distinctive outputs. Incorrect output can be easily identified. Internal errors are automatically detected and reported

3. Controllability: Controllability: The term "controllability" describes the degree to which an internal or external factor can influence the outcome of a process or system. The ability to influence a system's behaviour to produce the desired results is called "controllability," and it is utilised as a fundamental term in many different domains, such as engineering, control theory, and software development.

Software and hardware states and variables can be controlled directly by the test engineer. Tests can be conveniently specified, automated, and reproduced. "The better we can control the software, the more the testing can be automated and optimised." Some combination of input can be used to generate all possible outputs. I/O formats are consistent and structured

4. Decomposability: Decomposability is a term used in several disciplines, such as software engineering, systems analysis, and design, to describe how easily a complex system can be partitioned into simpler subsystems. It is easier to analyse, design, develop, and test these subsystems individually before reassembling them into the whole.

   "By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting." Independent modules are used to built the software system that can be independently tested. An essential principle in both software engineering and system architecture is decomposability. It fosters code reusability, allows for simultaneous development, and helps teams handle the complexity of large-scale systems. In general, well-decomposed systems are less of a hassle to keep up with and modify as needed.

5. Simplicity: There are several fields where the notion of simplicity is held in high esteem; some examples include user interface design, product design, software engineering, architecture, and communication. Users are more satisfied, mistakes and confusion are less likely, and productivity is increased. Keep the program as simple as possible."The less there is to test, the more quickly we can test it."

   The program should exhibit

   - functional simplicity (e.g., only the feature set is essential to meet requirements);
   - structural simplicity (e.g., modular architecture is used to limit the propagation of faults), and
   - code simplicity (e.g., a coding standard is followed so as to simplify inspection and maintenance).

6. Stability: "The fewer the changes, the fewer the disruptions to testing." Changes to the software are kept minimal and controlled when they do occur, and do not nullify existing tests. The software is recovered from failures

   To be stable is to be sturdy, secure, and unaffected by external influences. In fields as diverse as engineering, systems, economics, and social structures, this quality is highly prized. To be stable, a system must be resilient enough to keep working normally despite changes in its environment or its own internal dynamics.

   In many fields, stability plays a key role. It is crucial for the development of secure and trustworthy engineering structures and systems. In economics, it is crucial to the economy's long-term viability. It is essential in social systems because it guarantees the survival of groups and societies. In general, systems are more resilient and perform better when they are stable.

7. Understand ability: Technical documentation is instantly accessible, well organized, specific and detailed, and accurate. Changes to the design are communicated to testers.

   "The more information we have, the smarter we will test." Easily understood architectural design and the dependencies between internal, external, and shared components

   The term "undesirability" describes the condition of being unwelcome. It denotes something that is undesirable or useless. To put it simply, this word is used to indicate anything that does not meet with approval or approval.What is deemed unpleasant, however, can be both relative and situational. Something that is unfavourable in one setting or to one individual may be acceptable to another. As a result, the phrase is typically applied to targeted aims, tastes, or criteria.

Internal and External Views of Testing

- Tests can be conducted exhibiting that each function is fully operational while searching for errors in each function at the same time, knowing the specific function of each product. This test approach takes an external view and is termed black-box testing.

- Aware of the internal workings of a product, tests can be conducted to ensure that internal operations are performed according to specifications and all internal components have been properly checked. This approach requires an internal view and is referred as white-box testing
- Testing from the Inside: Examination of the tested software's underlying structures and logic is the main issue of the internal testing perspective. Finding bugs in software requires an examination of the source code, algorithms, data structures, and programme flow.The primary goal of internal testing is to verify the code's correctness in terms of functionality, adherence to programming standards, and suitable handling of inputs and scenarios.Unit testing, integration testing, and code reviews are all examples of methodologies used for in-house software quality assurance testing. Developers or a dedicated testing team who have access to the source code do these tasks.
- Testing from the Inside: Examination of the tested software's underlying structures and logic is the main issue of the internal testing perspective. Finding bugs in software requires an examination of the source code, algorithms, data structures, and programme flow. The primary goal of internal testing is to verify the code's correctness in terms of functionality, adherence to programming standards, and suitable handling of inputs and scenarios. Unit testing, integration testing, and code reviews are all examples of methodologies used for in-house software quality assurance testing. Developers or a dedicated testing team who have access to the source code do these tasks.
- Tests from the Outside Looking In: Emphasis: This testing approach considers the software from the client's or user's point of view. The emphasis is on the software's capabilities as they are perceived by the user. The main purpose of an external test is to ensure that the programme works as expected and gives a positive user experience.
- Tools Acceptance testing, system testing, user acceptance testing (UAT), usability testing, and beta testing are all examples of external testing methods. These tasks are often carried out by end users or external testers who have no access to the original code. Benefits: Users can rest easy knowing that their software has been put through

rigorous external testing. It aids in discovering where the software deviates from the expected behaviour.

The white box testing technique does the following,

1) Ensures that all the different path the code execution can take place is tested at least once.
2) Executes all the possible combination of logical decision making in the code.
3) Execute all loops at their boundaries and within their operational bounds.
4) Ensures the validity of internal data structure by exercising them.

Now let's discuss various white box testing techniques:

Basis path testing

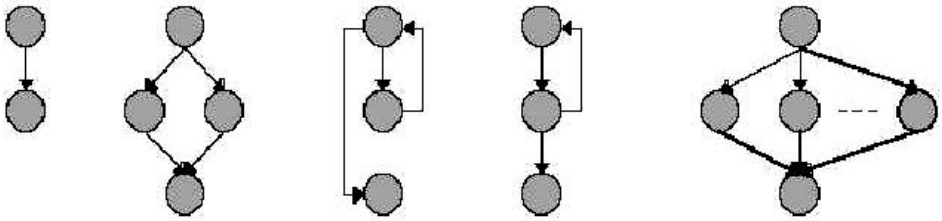
Basis path testing is a white-box testing technique first proposed by Tom McCabe. This method helps the test case designer to compute the measure of logical complexity of the design and use this measure as the basis for fixing the set of execution paths. The test cases developed through basis path testing method are guaranteed to execute each statement of the program at least once from the coverage point of view,

Flow graph notation

A simple flow graph or control flow graph is a graph based concept. It specifies a control flow of a program, helps to understand program. Flow graph is combination of nodes and edges, nodes are basic blocks represent computation or condition; edges are connecting lines between two nodes (i.e. flow of control).Each node that represent condition is called predicate node.

Cyclomatic complexity is a software metric, can be calculated by using control flow graph, this is discussed below in this section.

Notation for representing control flow is shown in figure 12.1.



Sequence If While Until Case

Fig. 12.1: Flow Graph Notations

On a flow graph:

- Arrows called *edges* represent flow of control
- Circles called *nodes* represent one or more actions.
- Areas bounded by edges and nodes are called *regions*.
- A *predicate* node is a node containing a condition

Any procedural design can be translated into a flow graph. Note that compound Boolean expressions at tests generate at least two predicate node and additional arcs.

Cyclomatic complexity

Cyclomatic complexity is software metric used to measure the logic complexity of source code/program. If the cyclomatic complexity is denoted in terms of basis path testing, it is derived from the number of linearly independent paths through the source code of program/ function. An independent path is any path through the program or function that introduces at least one new set of processing statements or a new condition. Independent path is a direct measure of complexity of application. Using flow graph an independent path is at least on edge that has not been traversed before in other paths.

For example, a set of independent paths for the flow graph illustrated in figure 12.2 is given below:

Path 1: 1-11.

Path 2: 1-2-3-4-5-10-1-11

Path 3: 1-2-3-6-8-9-10-1-11

Path 4: 1-2-3-6-7-9-10-1-11

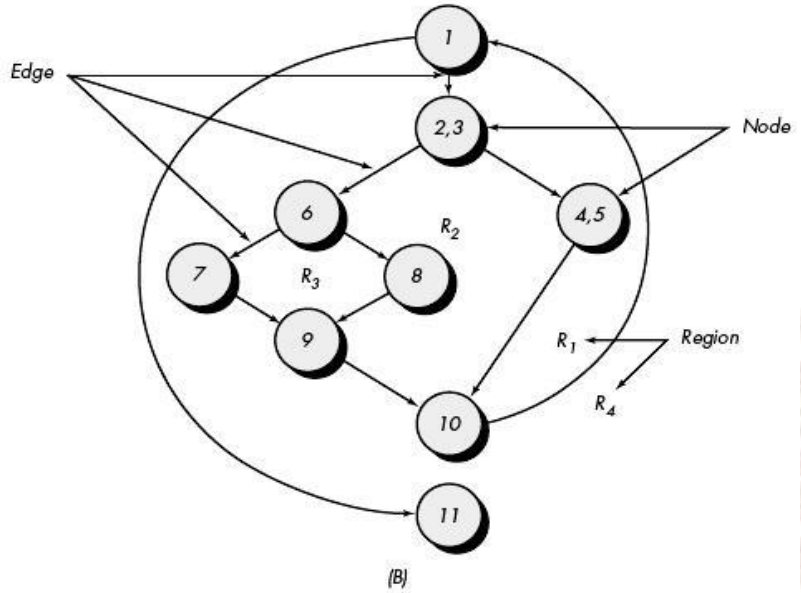


Fig. 12.2: An Example Flow Graph

Note that each new path introduces a new edge. The path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 is not considered to be an independent path because it is simply a combination of already specified paths and does not traverse any new edges. Cyclomatic Complexity is computed in one of three ways:

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.
2. Cyclomatic complexity, *V(G),* for a flow graph, G, is defined as *V(G) = E –N + 2*. Where E is the number of flow graph edges, N is the number of flow graph nodes.
3. Cyclomatic complexity, *V(G),* for a flow graph, *G*, is also defined as *V(G) = P + 1*. Where P is the number of predicate nodes contained in the flow graph G.

Referring again to the flow graph in figure 12.2, the cyclomatic complexity can be computed using each of the algorithms just noted:

1. The flow graph has four regions.
2. V (G) = 11 edges – 9 nodes + 2 = 4.

3. V (G) = 3 predicate nodes + 1 = 4.

Therefore, the cyclomatic complexity of the flow graph in figure 12.2 is 4.

Control structure testing

Control structure testing as name suggests focusses on testing the control statements or loops in the software. The different types of control structure testing are listed as follows

Loop testing:

Loops are fundamental to many algorithms and need thorough testing. There are four different classes of loops: simple, concatenated, nested, and unstructured (See Figure 12.3).

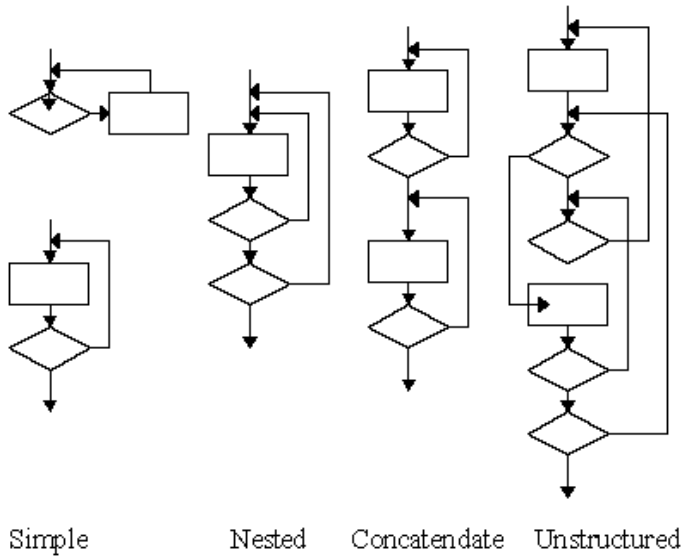


Fig. 12.3: Loop structures

- ***Simple loops***, Suppose n is the number of passes allowed through the loop, the testing would involve
  - Skip loop entirely
  - test with only one pass o test with the passes
  - try with m passes where m<n.
  - (n-1), n, and (n+1) passes through the loop.

- ***Nested loops :*** For nested loops test as follows
  - Start with inner loop. Set all other outer loops with terminal conditions attaching minimum value.
  - Conduct simple loop testing on inner loop. o extend towards outer loops
  - Complete the testing continuing in same fashion covering all the loops.
- ***Concatenated loops***
  - If the loops are not interdependent execute the test as simple loops. o If the loops are dependent then carry out the testing as nested loops.
- ***Unstructured loops***
  - Do not test this code, instead redesign the solution

Advantages of White Box testing

i) Since the tester has the pre-requisite knowledge on the software, the testing becomes more effective as tester can define meaningful set of test data and test conditions.
ii) White box testing helps in code optimization
iii) It helps in removal of un-necessary code from the application as these can cause some unknown defects in the long run of the application.

Disadvantages of White Box testing:

i) For performing white box testing the tester should have the knowledge of the code and internal structure of the application. This prerequisite increases the cost of testing.
ii) Though white box testing intends to test the internal structure of the application, it is impossible to look at every line of code and various code combinations that execution can take place.

Testing tools

Following are some of the popular Software Testing Tools:

- Win Runner
- Load Runner

- Test Director
- Silk Test
- Test Partner

SELF-ASSESSMENT QUESTIONS -3

7. White box testing is a test case design method that uses the control structure of the procedural design to derive test cases. (True / False)
8. _____ is software metric that provides a quantitative measure of the logical complexity of a program.
9. Which one of the following is a Software Testing tool? (Pick right option)

a) Rational Rose
b) Oracle
c) Win Runner
d) Linux

6. SUMMARY

Let's recapitulate important points discussed in this unit:

- The primary objective for test case design is to derive a set of tests that have the highest likely hood for uncovering errors in the software. To accomplish this objective, two different categories of test case design techniques are used: White–Box testing and Black–Box testing.
- White–box testing focuses on the program control structure. Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been exercised.
- Basis path testing, a white–box technique, makes use of program graphs (or graph matrices) to derive the set of linearly independent tests that will ensure coverage. Condition and data flow testing further exercise program logic, and loop testing complements other white–box techniques by providing a procedure for exercising loops of varying degrees of complexity.
- Black–box testing, on the other hand, broadens our focus and might be called "testing in large". Black–box tests are designed to validate functional requirements without regard to the internal workings of a program.
- Black–box testing techniques focus on the information domain of the software, deriving test cases by partitioning the input and output domain of a program in a manner that provides thorough test coverage.

7. TERMINAL QUESTIONS

1. Explain the need for Software Testing.
2. What are the two broad types of Software Testing?
3. What do you mean by Black Box Testing? Why it is called so?
4. Briefly explain the significance of White Box Testing.

8. ANSWERS

Self Assessment Questions

1. True
2. Scalability Testing
3. a) 40%
4. False
5. Boundary Value Analysis
6. b) Cause-effect graphing
7. True
8. Cyclomatic complexity
9. c) Win Runner

Terminal Questions

1. Software testing is the process of running through the application or a software product with the intention of uncovering the errors. It is the mechanism by which one can ascertain that the product meets its intended specifications. (Refer Section 2 for detail)
2. Software testing consists of several subcategories, each of which is done for different purposes, and often – using different techniques. Software testing categories include: Functionality Testing, Forced Error Testing, Compatibility Testing, Performance Testing, Scalability Testing, Stress Testing, Usability Testing, Application Security Testing etc. (Refer Section 3)
3. Black Box testing focusses mainly on the functional testing. It tries to derive the inputs such that all the functional area of the product being tested is covered. (Refer Section 4)
4. White box testing is a test case design method that uses the control structure of the procedural design to derive test cases. (Refer Section 5)