Software Engineering | Unit – 4

Q1. What is Testing?

“Testing is the process of executing a program with the intent of finding errors.”

Q2. Why should we do testing?

Although software testing is itself an expensive activity, yet launching of software without testing may lead to cost potentially much higher than that of testing, specially in systems where human safety is involved. In the software life cycle the earlier the errors are discovered and removed, the lower is the cost of their removal.

Q3. What is fault?

A fault is the representation of an error, where representation is the mode of expression, such as narrative text, data flow diagrams, ER diagrams, source code etc. Defect is a good synonym for fault.

Q4. What is failure?

A failure occurs when a fault executes. A particular fault may cause different failures, depending on how it has been exercised.

Q5. What is a Test Case?

Test case describes an input description and an expected output description.

Test Case ID	
Section-I (Before Execution)	Section-II (After Execution)
Purpose :	Execution History:
Pre condition: (If any)	Result:
Inputs:	If fails, any possible reason (Optional);
Expected Outputs:	Any other observation:
Post conditions:	Any suggestion:
Written by:	Run by:
Date:	Date:

Fig. 2: Test case template

The set of test cases is called a test suite. Hence any combination of test cases may generate a test suite.

Q6. What is Verification?

Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Q7. What is Validation?

Validation is the process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements .
Testing= Verification+Validation

Q8. What is Acceptance Testing?

The term Acceptance Testing is used when the software is developed for a specific customer. A series of tests are conducted to enable the customer to validate all requirements. These tests are conducted by the end user / customer and may range from adhoc tests to well planned systematic series of tests.

Q9. What is Functional Testing?

Functional testing is defined as a type of testing that verifies that each function of the software application works in conformance with the requirement and specification. This testing is not concerned with the source code of the application. Each functionality of the software application is tested by providing appropriate test input, expecting the output, and comparing the actual output with the expected output.

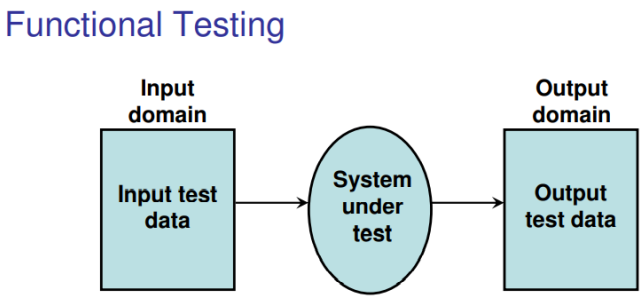


Fig. 3: Black box testing

Q10. What is Boundary Value Analysis?

Boundary Value Analysis is based on testing the boundary values of valid and invalid partitions. The behavior at the edge of the equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.

It checks for the input values near the boundary that have a higher chance of error. Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.

Consider a program with two input variables x and y. These input variables have specified boundaries as:

$$a \le x \le b$$
$$c \le y \le d$$

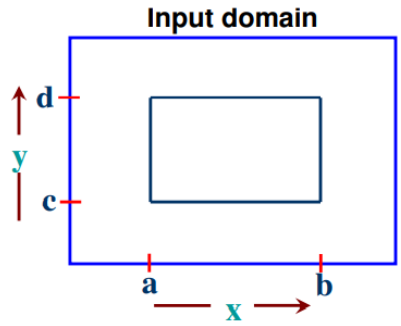


Fig.4: Input domain for program having two input variables

Q11. What is Robustness Testing?

It is nothing but the extension of boundary value analysis. Here, we would like to see, what happens when the extreme values are exceeded with a value slightly greater than the maximum, and a value slightly less than minimum. It means, we want to go outside the legitimate boundary of input domain. This extended form of boundary value analysis is called robustness testing.

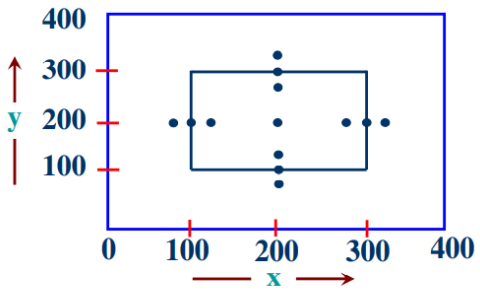


Fig. 8.6: Robustness test cases for two variables x and y with range [100,300] each

Q12. What is Worst Case testing?

Boundary Value analysis uses the critical fault assumption and therefore only tests for a single variable at a time assuming its extreme values. By disregarding this assumption we are able to test the outcome if more than one variable were to assume its extreme value. In an electronic circuit this is called Worst Case Analysis. In Worst-Case testing we use this idea to create test cases.

Q13. What is Equivalence Class Testing?

Equivalence Class Testing, which is also known as **Equivalence Class Partitioning (ECP)** and **Equivalence Partitioning**, is an important software testing technique used by the team of testers for grouping and partitioning of the test input data, which is then used for the purpose of testing the software product into a number of different classes.

These different classes resemble the specified requirements and common behaviour or attribute(s) of the aggregated inputs. Thereafter, the test cases are designed and created based on each class attribute(s) and one element or input is used from each class for the test execution to validate the software functioning, and simultaneously validates the similar working of the software product for all the other inputs present in their respective classes.

Two steps are required to implementing this method:

1. The equivalence classes are identified by taking each input condition and partitioning it into valid and invalid classes. For example, if an input condition specifies a range of values from 1 to 999, we identify one valid equivalence class [1<item<999]; and two invalid equivalence classes [item<1] and [item>999].

2. Generate the test cases using the equivalence classes identified in the previous step. This is performed by writing test cases covering all the valid equivalence classes. Then a test case is written for each invalid equivalence class so that no test contains more than one invalid class. This is to ensure that no two invalid classes mask each other.

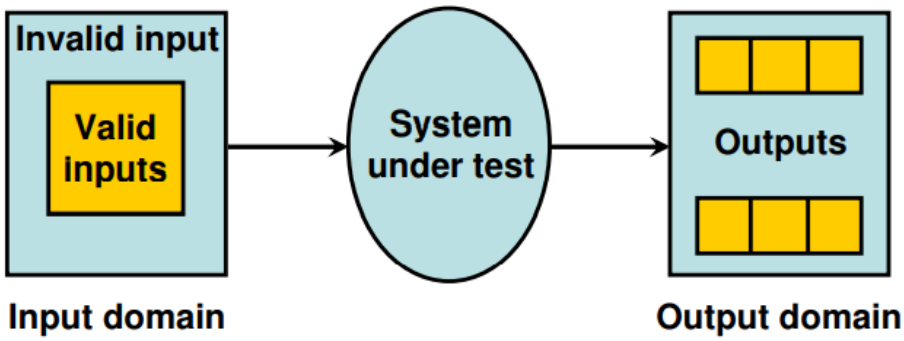


Fig. 7: Equivalence partitioning

Most of the time, equivalence class testing defines classes of the input domain. However, equivalence classes should also be defined for output domain. Hence, we should design equivalence classes based on input and output domain.

Q14. What is Decision Table Testing?

Decision tables are used in various engineering fields to represent complex logical relationships. This testing is a very effective tool in testing the software and its requirements management. The output may be dependent on many input conditions and decision tables give a tabular view of various combinations of input conditions and these conditions are in the form of True(T) and False(F). Also, it provides a set of conditions and its corresponding actions required in the testing.

Condition Stub		Entry						
C_1		True				False		
C_2		True		False		True		False
C_3		True	False	True	False	True	False	---
Action Stub	a_1	X	X			X		
	a_2	X		X			X	
	a_3		X			X		
	a_4				X		X	X

Table 2: Decision table terminology

In software testing, the decision table has 4 parts which are divided into portions and are given below :

1. **Condition Stubs :** The conditions are listed in this first upper left part of the decision table that is used to determine a particular action or set of actions.
2. **Action Stubs :** All the possible actions are given in the first lower left portion (i.e, below condition stub) of the decision table.
3. **Condition Entries :** In the condition entry, the values are inputted in the upper right portion of the decision table. In the condition entries part of the table, there are multiple rows and columns which are known as Rule.

4. **Action Entries :** In the action entry, every entry has some associated action or set of actions in the lower right portion of the decision table and these values are called outputs.

Q15. What is Cause Effect Graphing Technique?

Cause Effect Graphing based technique is a technique in which a graph is used to represent the situations of combinations of input conditions. The graph is then converted to a decision table to obtain the test cases. Cause-effect graphing technique is used because boundary value analysis and equivalence class partitioning methods do not consider the combinations of input conditions. But since there may be some critical behaviour to be tested when some combinations of input conditions are considered, that is why cause-effect graphing technique is used.

The basic notation for the graph is shown in fig. 8

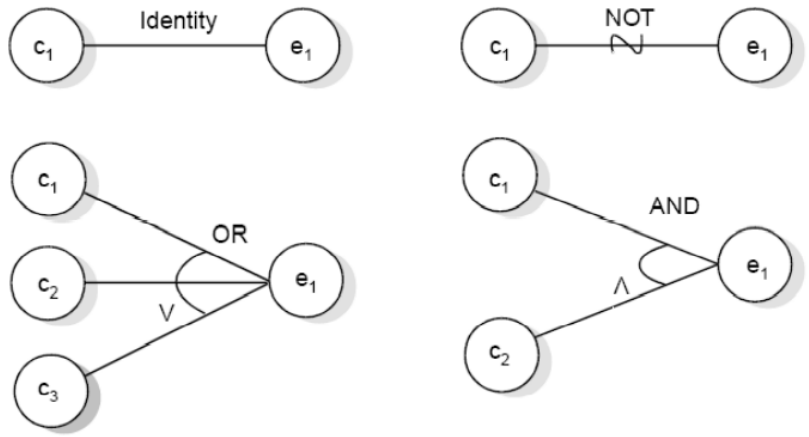
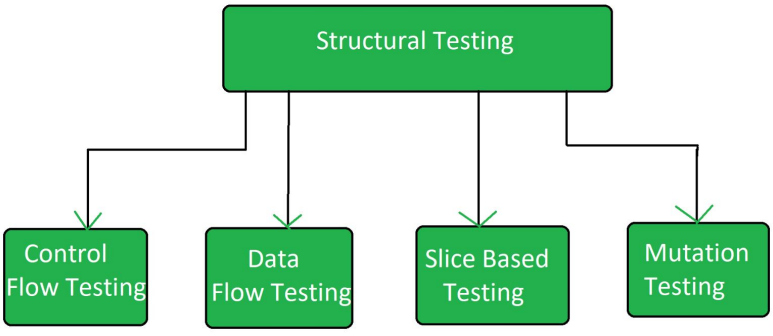


Fig.8. 8 : Basic cause effect graph symbols

Q16. What is 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.



Q17. What is Path 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.

This type of testing involves:
1. generating a set of paths that will cover every branch in the program.
2. finding a set of test cases that will execute every path in the set of program paths

Q18. What is 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.

The control flow of a program can be analysed using a graphical representation known as flow graph. The flow graph is a directed graph in which nodes are either entire statements or fragments of a statement, and edges represents flow of control.

Q19. What is 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.

i. Statements where variables receive values.
ii. Statements where these values are used or referenced.

As we know, variables are defined and referenced throughout the program. We may have few define/ reference anomalies:

i. A variable is defined but not used/ referenced.
ii. A variable is used but never defined.
iii. A variable is defined twice before it is used.

Definitions

The definitions refer to a program P that has a program graph G(P) and a set of program variables V. The G(P) has a single entry node and a single exit node. The set of all paths in P is PATHS(P)

(i) **Defining Node:** Node n ∈ G(P) is a defining node of the variable $v \in V$, written as DEF (v, n), if the value of the variable v is defined at the statement fragment corresponding to node n.

(ii) **Usage Node:** Node n ∈ G(P) is a usage node of the variable $v \in V$, written as USE (v, n), if the value of the variable v is used at statement fragment corresponding to node n. A usage node USE (v, n) is a predicate use (denote as p) if statement n is a predicate statement otherwise USE (v, n) is a computation use (denoted as c).

(iii) **Definition use:** A definition use path with respect to a variable v (denoted du-path) is a path in PATHS(P) such that, for some $v \in V$, there are define and usage nodes DEF(v, m) and USE(v, n) such that m and n are initial and final nodes of the path.

(iv) **Definition clear :** A definition clear path with respect to a variable v (denoted dc-path) is a definition use path in PATHS(P) with initial and final nodes DEF (*v*, m) and USE (*v*, n), such that no other node in the path is a defining node of *v*.

The du-paths and dc-paths describe the flow of data across source statements from points at which the values are defined to points at which the values are used. The du-paths that are not definition clear are potential trouble spots.

Hence, our objective is to find all du-paths and then identity those du-paths which are not dc-paths. The steps are given in Fig. 27. We may like to generate specific test cases for du-paths that are not dc-paths.

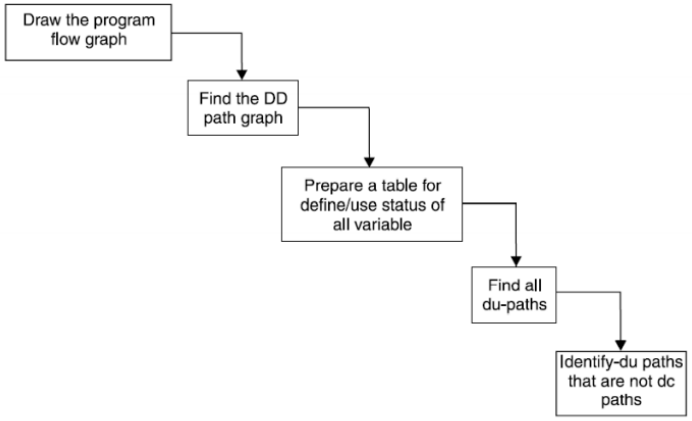


Fig. 27 : Steps for data flow testing

Q20. What is 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.

Mutation testing is a fault based technique that is similar to fault seeding, except that mutations to program statements are made in order to determine properties about test cases. it is basically a fault simulation technique.

Multiple copies of a program are made, and each copy is altered; this altered copy is called a mutant. Mutants are executed with test data to determine whether the test data are capable of detecting the change between the original program and the mutated program.

Q21. Difference between is Unit Testing and Integration Testing.

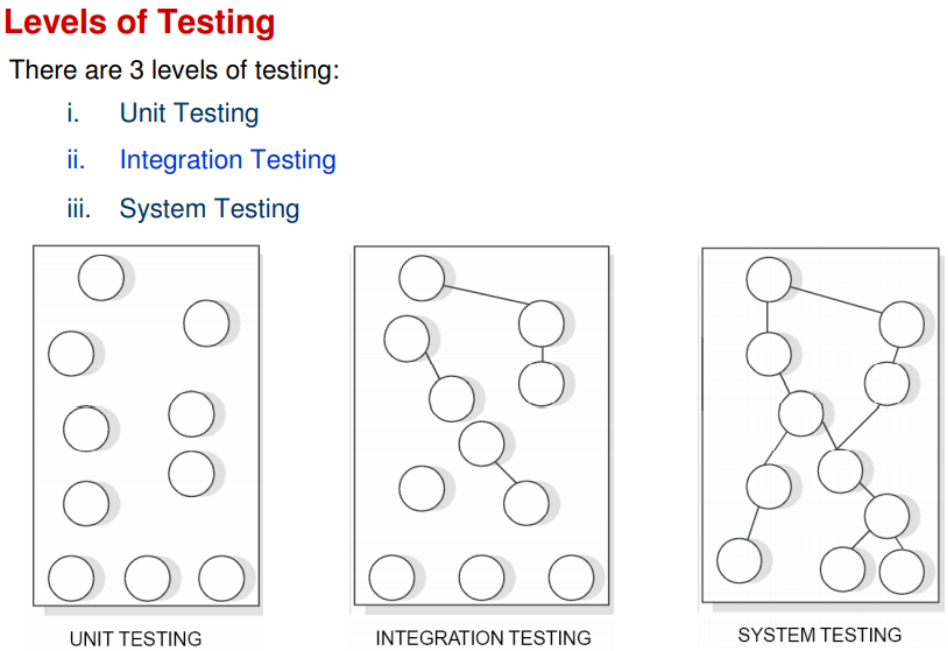
S. No.	Unit Testing	Integration Testing
1.	In unit testing, each module of the software is tested separately.	In integration testing, all modules of the software are tested combined.
2.	In unit testing tester knows the internal design of the software.	Integration testing doesn't know the internal design of the software.
3.	Unit testing is performed first of all testing processes.	Integration testing is performed after unit testing and before system testing.
4.	Unit testing is white box testing.	Integration testing is black box testing.
5.	Unit testing is performed by the developer.	Integration testing is performed by the tester.
6.	Detection of defects in unit testing is easy.	Detection of defects in integration testing is difficult.
7.	It tests parts of the project without waiting for others to be completed.	It tests only after the completion of all parts.
8.	Unit testing is less costly.	Integration testing is more costly.
9.	Unit testing is responsible to observe only the functionality of the individual units.	Error detection takes place when modules are integrated to create an overall system.
10.	Module specification is done initially.	Interface specification is done initially.
11.	The proper working of your code with the external dependencies is not ensured by unit testing.	The proper working of your code with the external dependencies is ensured by integration testing.
12.	Maintenance is cost effective.	Maintenance is expensive.
13.	Fast execution as compared to integration testing.	Its speed is slow because of the integration of modules.
14.	Unit testing results in in-depth exposure to the code.	Integration testing results in the integration structure's detailed visibility.

Q22. Difference Between Unit Testing and System Testing

Aspects	Unit Testing	System Testing
Scope of testing	Tests individual units or components of the software application in isolation	Tests the complete system or software application
Timing of testing	Typically automated and performed by developers during the coding phase	Typically manual and performed by independent testers after the completion of integration testing
Focus of testing	Focuses on verifying the functionality of each unit or component of the software application	Focuses on verifying the functionality of the system as a whole
Testing environment	Tests are executed in a controlled environment, typically using mock objects or test doubles to isolate the unit being tested	Tests are executed in a production-like environment to simulate real-world usage scenarios

Aspects	Unit Testing	System Testing
Testing frameworks	Uses unit test frameworks such as JUnit, NUnit, or PHPUnit to automate the testing process	Uses system testing frameworks such as Selenium, HP UFT, or IBM Rational Functional Tester to automate the testing process
Purpose of testing	Enables early detection and elimination of defects in the code	Enables validation of the entire software application to ensure it meets the specified requirements
Skills required	Requires a solid understanding of the code and the ability to write effective test cases	Requires a comprehensive understanding of the software application and the ability to write effective test cases
Feedback	Provides fast feedback on the quality of individual units or components	Provides a comprehensive understanding of the quality and readiness of the entire software application
Execution	Typically executed by developers in a continuous integration and delivery (CI/CD) pipeline	Typically executed by independent testers in a separate testing environment
Types of testing	Tests for both functional and non-functional aspects of the software application	Tests for both functional and non-functional aspects of the software application, including performance, security, and usability testing

Q23. What are the levels of testing?



Q24. What is Debugging?

The goal of testing is to identify errors (bugs) in the program. The process of testing generates symptoms, and a program's failure is a clear symptom of the presence of an error. After getting a symptom, we begin to investigate the cause and place of that error. After identification of place, we examine that portion to identify the cause of the problem. This process is called debugging.

Q25. What is Induction Debugging Approach?

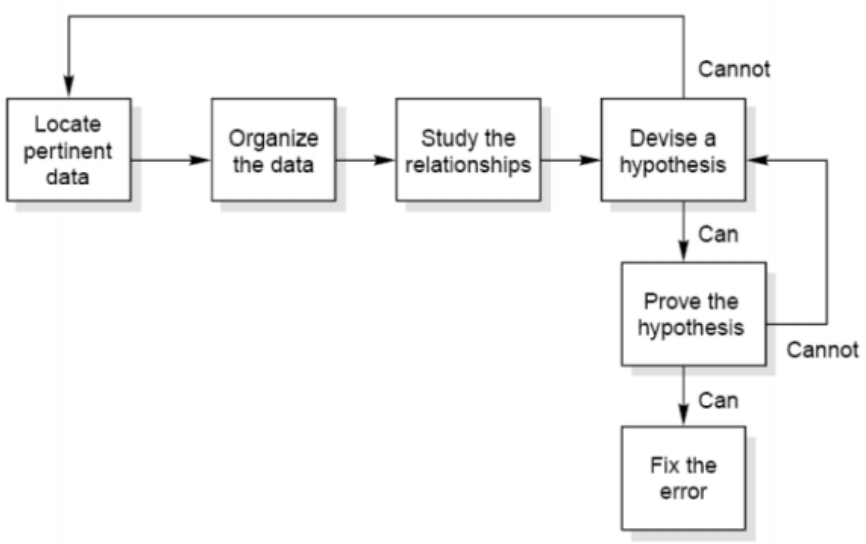


Fig. 32 : The inductive debugging process

Q26. What is Deduction Debugging Approach?

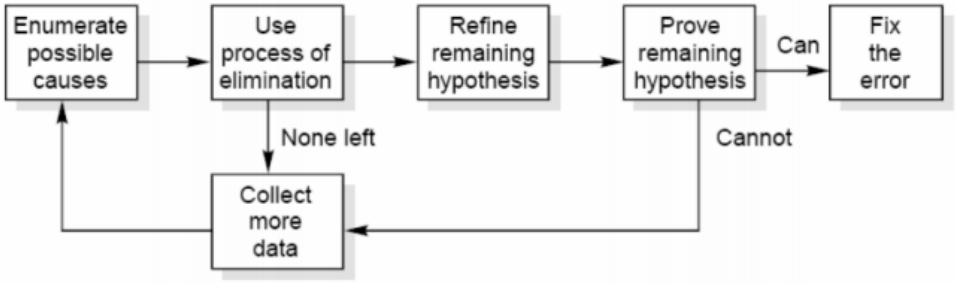


Fig. 33 : The inductive debugging process

Q27. What are Testing Tools?

One way to improve the quality & quantity of testing is to make the process as pleasant as possible for the tester. This means that tools should be as concise, powerful & natural as possible.
The two broad categories of software testing tools are :

- Static
- Dynamic

Q28. What are the types of testing tools?

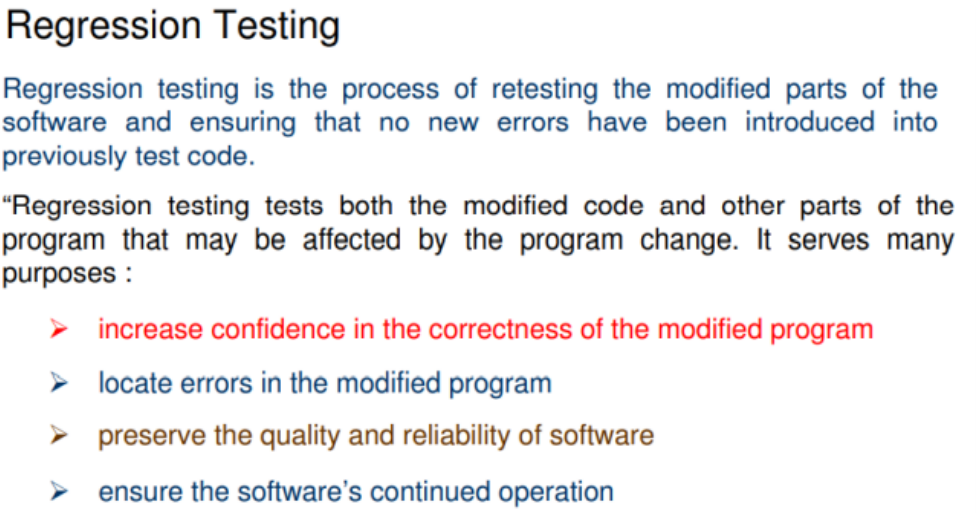
Static Test Tools: Static test tools are used to work on the static testing processes. In the testing through these tools, the typical approach is taken. These tools do not test the real execution of the software. Certain input and output are not required in these tools. Static test tools consist of the following:

- **Flow analyzers:** Flow analyzers provides flexibility in the data flow from input to output.
- **Path Tests:** It finds the not used code and code with inconsistency in the software.
- **Coverage Analyzers:** All rationale paths in the software are assured by the coverage analyzers.
- **Interface Analyzers:** They check out the consequences of passing variables and data in the modules.

Dynamic Test Tools: Dynamic testing process is performed by the dynamic test tools. These tools test the software with existing or current data. Dynamic test tools comprise the following:

- **Test driver:** The test driver provides the input data to a module-under-test (MUT).
- **Test Beds:** It displays source code along with the program under execution at the same time.
- **Emulators:** Emulators provide the response facilities which are used to imitate parts of the system not yet developed.
- **Mutation Analyzers:** They are used for testing the fault tolerance of the system by knowingly providing the errors in the code of the software.

Q29. What is Regression Testing?



Q30. Difference between Development and Regression Testing?

- Development Testing Versus Regression Testing

Sr. No.	Development testing	Regression testing
1.	We create test suites and test plans	We can make use of existing test suite and test plans
2.	We test all software components	We retest affected components that have been modified by modifications.
3.	Budget gives time for testing	Budget often does not give time for regression testing.
4.	We perform testing just once on a software product	We perform regression testing many times over the life of the software product.
5.	Performed under the pressure of release date of the software	Performed in crisis situations, under greater time constraints.

Q31. Difference between Functional Testing and Structural Testing?

Structural Testing	**Functional Testing**
This test evaluates the code structure or internal implementation of the code.	This test checks whether the software is functioning in accordance with functional requirements and specifications.
It is also known as white-box or clear-box testing as thorough knowledge and access of the code is required.	It is also known as black-box testing as no knowledge of the internal code is required.
Finds errors in the internal code logic and data structure usage.	It ensures that the system is error-free.
It does not ensure that the user requirements are met.	It is a quality assurance testing process ensuring the business requirements are met.
Performed the entire software in accordance with the system requirements.	Functional testing checks that the output is given as per expected.
Testing teams usually require developers to perform structural testing.	A QA professional can simply perform this testing.
Perform on low-level modules/software components.	The functional testing tool works on event analysis methodology.
It provides information on improving the internal structure of the application.	It provides information that prevents business loss.
Structural testing tools follow data analysis methodology.	Functional testing tool works on event analysis methodology.
Writing a structural test case requires understanding the coding aspects of the application.	Before writing a functional test case, a tester is required to understand the application’s requirements.
It examines how well modules communicate with one another.	It examines how well a system satisfies the business needs or the SRS.