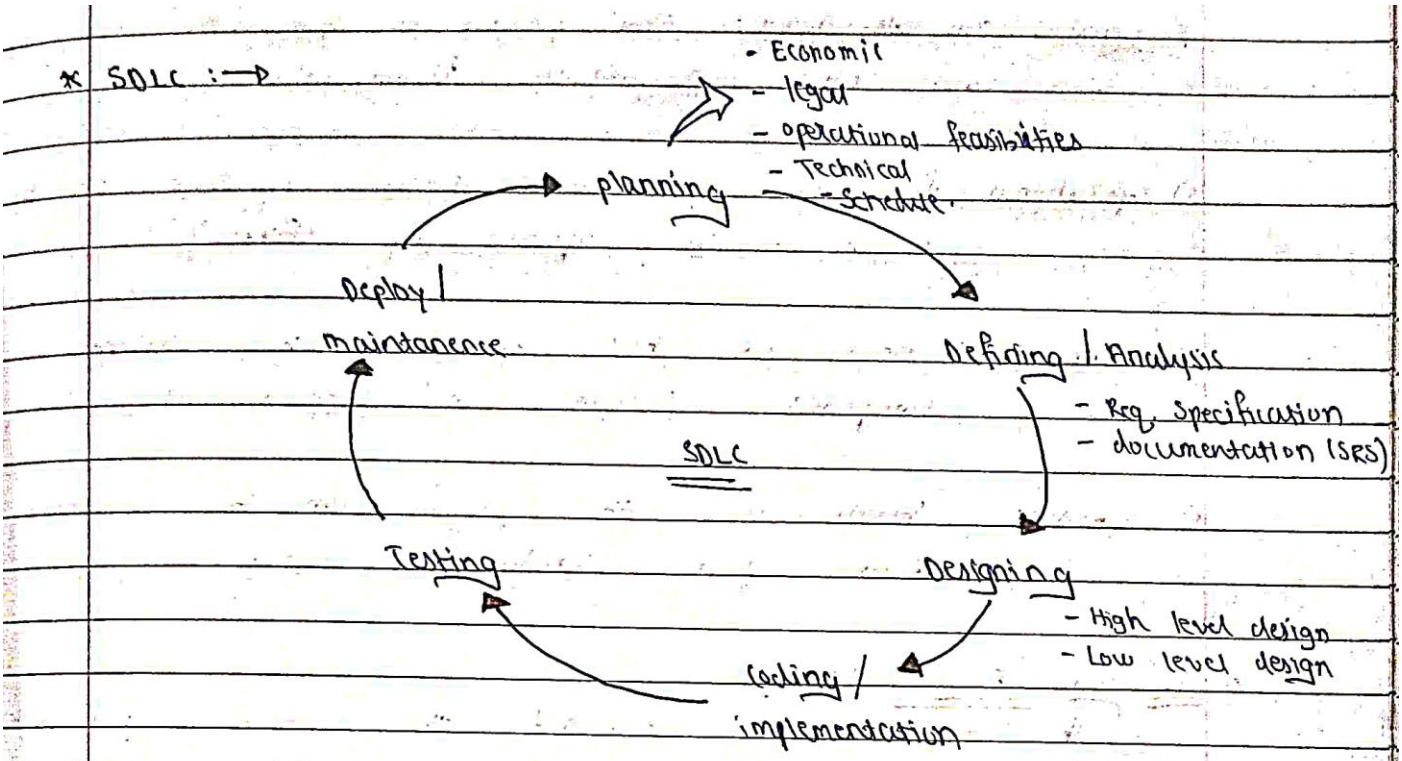
STQA CAT1 ANSWERS

1. a) What is SDLC? Describe in detail.



① SDLC stands for Software Development life cycle. It describe the sequence of phases or steps to develop any software.

② In simple words, "entire lifetime of s/w from begining to ending"

a) planning :→

In involves analysis of problem and it is most important and fundamental stages. If a s/w company any customer comes the the problem and it is analysed that it is feasible or not to solve the customers problem.

Here estimation of rough idea of the resource requirements as well as the estimated time for the completion is given. It is performed by the senior member of the team. There are mainly five types of the feasibility checks.

1) Economic : can we complete the project within the budget or not.

2) Legal : can we handle this project as per the cyber law and other regulatory frameworks.

3) operational feasibility : can we create operation o/c to which it is expected by clients ?

4) Technical : need to check whether the current computer system can support the s/w.

5) Schedule : beside that the project can be completed within given schedule or not.

b) Defining / analysis :→

The aim is to understand the exact requirements of the customer, it consist of two distinct activities

① Requirement gathering and analysis :

The goal is to collect all the revelent info. from customer regarding the product to be developed so that the incompleteness and inconsistent are removed.

② Requirement specification / documentation :

After all the ambiguities, inconsistencies and incompleteness have been resolved and all the requirements are properly understood, the requirement specification work.

Requirement document is very imp. activity and it is systematically organised into a s/w requirement specification (SRS)

c) Design document :→

The goal is the to transform the requirement specified in the SRS document into a structure that is suitable for implementation in some programming languages.

We design the architecture of the system using SRS document which means skeleton or a blueprint of the project.

Two kinds of design documents are developed in this phase :

① HLD :

High level design. It's a complete architecture design / diggram.

② LLD :

Low level design. In this complete details / design and listing of error messages are given.

d) Coding :→

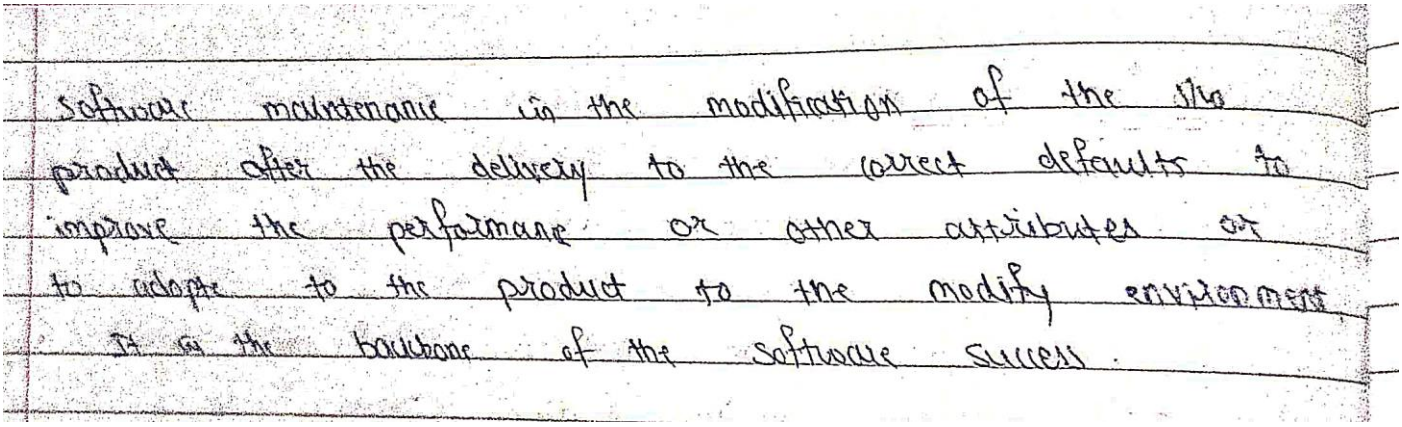
Developers starts building the entire system by writing code using the chosen programming language. The system first is developed in small programs called as units / module / modules.

In coding phase, the task are divided into units / modules and are assigned to various developers.

It is the longest phase of the SDLC process. Developers of all the levels including Senior, Juniors, freshers are involved in this phase.

e) Testing :→

It is the important part of the project as it helps to improve the quality of the projects. After the development of s/w there can be many defects in the system or system may not be working as they are expected.



Deployment and Maintenance Of Product:

After detailed testing, the conclusive product is released in phases as per the organization's strategy. Then it is tested in a real industrial environment. Because it is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the product's supervision.

1. b) Difference between Verification and Validation.

Verification	Validation
① It includes checking documents, design, codes and programs	① In validation, testing and validating the actual product.
② It is static testing	② It is the dynamic testing.
③ It does not include the execution of the code	③ It includes the execution of the code
④ Methods used in verification are reveiws, walkthroughs, inspection and desk checking	④ methods in validation are black box testing, white box testing & Non-functional testing.
⑤ It checks whether the s/w conforms the specification or not	It checks whether the s/w meets the requirements and expectation of a customer or not.
⑥ It can find bugs in early stage of development	⑥ It can only find the bugs that could not be found by verification process
⑦ The goal of verification is application and s/w architecture and specification	⑦ The goal of validation is the actual product

	Verification	Validation
(8)	Quality assurance team does verification	(8) Validation is executed on s/w code with the help of testing team
(9)	It comes before validation	(9) It comes after verification.
(10)	It consist of checking of documents / files and is perform by human	(10) It consist of execution of program and it is performed by computers.

2. a) Explain in detail about the following terminologies in testing:
Error, Fault, Failure

* Terminologies in Testing :→
Error, fault, failur, verification, validation.

① Error :→
The problem in code leads to errors, which means that a mistake can occur due to the developer coding error or the developer misunderstood the requirements or the requirement not defined correctly. The developers use the term error.

② Fault :→
The fault may occur in SW because it has not added the code for fault tolerance, making an application act up. Following reasons:
① lack of resources
② invalid step
③ inappropriate data definition
can lead to a fault to occur in program.

③ Failur :→
many defects leads to the failur which means that a bug specifies a fatal issue in SW or in its module, which makes the system unresponsive or broken. In other words, we can say if the end user detects and issues in the product then that particular issue called the failur.
ex.
In a bank application if account transfer module is not working for end users when the end user tries to transfer money, Submit button is not working. Hence this is failur.

2. b) What is test case? Design the test case to for the scenario:
"Check Login Functionality"

4. b) What is Test Case Scenario? Explain in detail with suitable example.

* Test cases:-
test steps
test data
precondition
post condition
test scenario

Test case is a set of action executed to verify the particular feature or functionality of our sw application. A test case contains test steps, test data, precondition, post condition and test scenarios developed for specific test scenario to verify any requirement. The test case includes specific variables or condition using which a testing engineer can compare expected and actual results to determine

whether a the product is functioning as per the requirements of the customer.

For a test scenario, check login functionality there are many possible test cases.

Test case 1: →

check results on entering valid user id & password

Test case 2: →

check results on entering invalid user id & password.

Test case 3: →

check response when userid is empty and login button is pressed, and many more

The formate of a standard test case:

Test case ID	Test case description	Test steps	Test Data	Expected Result	Actual Result	Pass / fail
TU01	check cust. login with valid data	1. website http://www.guru.com 2. userid 3. enter password 4. click to submit	userid = guru99 password = pass99	user should log in into an app	As expected	Pass
TU02	Invalid: check cust. with invalid data	——"——	user id = guru199 password: pass99	user should Not login into an App	As expected	pass
TU02	Empty	——"——	user id: empty password: passgg	user should Not login into an Application	As expected	pass

This entire test may be created in word, excel or any other test management tol

3. a) What are the different practices for designing good test case? Explain in detail.

3. b) What do you mean by Test Case Management? Enlist any two Test Case Management Tools.

A test case is a document, which has a set of test data, preconditions, expected results and postconditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test management in the field of software testing refers to the process of planning, organizing, and controlling the testing activities and resources of a project. Software testing management involves creating and executing test strategies, defining test plans and/or test cases, and managing defects to ensure that the software meets the desired quality standards.

Effective test management is crucial for the success of a software project and involves collaboration with all product stakeholders to ensure that the testing process is streamlined and efficient.

Test Case acts as the starting point for the test execution, and after applying a set of input values, the application has a definitive outcome and leaves the system at some end point or also known as execution postcondition.

Test case management is the process of managing testing activities to ensure high-quality and end-to-end testing of software applications. To deliver a high-quality software application, the method entails organizing, controlling, and ensuring traceability and visibility of the testing process. In addition, it ensures that the software testing process is continuous as per your plan.

TESTRAIL AND TESTGEAR ARE TOOLS

4. a) What are Test Oracles? Explain in detail.

* Test oracles :→

It is a mechanism different from program itself that can be used to test the accuracy of the program output for test cases. Conceptually we can consider testing a process in which test cases are given for testing and the program under test. The output of the two then compares to determine whether the program behaves correctly for the test cases. Shown in diagram below.

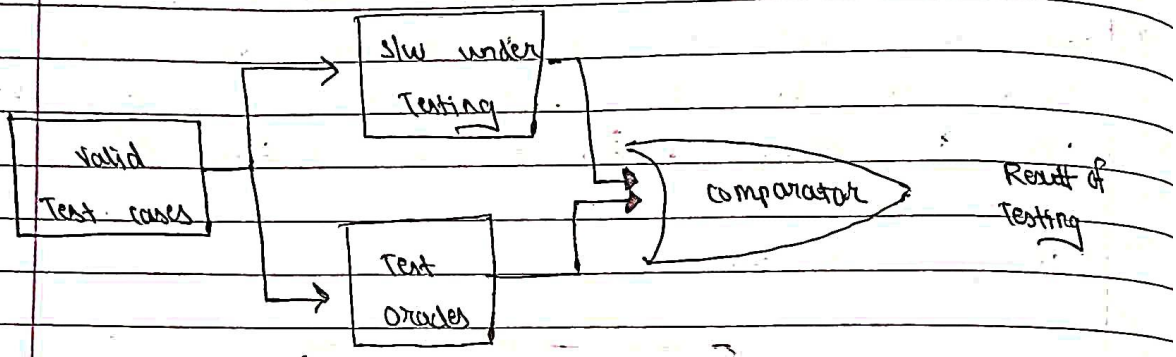


Fig: Test oracles and testing

Testing oracles are required for testing. Ideally we want an automated oracle which always gives the "correct" answer. However often 'oracles' are human being who mostly calculated by hand what the o/p of program should be. As it is often very difficult to determine whether the behaviour corresponds to the expected behaviour, are "human deities" may make mistakes.

Consequently, when there is diseri between program and result, we must verify the result produced by the oracle before declaring that there is a defect in the result.

The human oracle typically used the program specifications to decide what the correct behaviour of the program should be. To help oracle determine the

correct behaviour it is imp that the behaviours of the system or the component is explicitly specified and the specification itself be error-free.

There are some systems where oracle are automatically generated with such oracles, we are assure that the o/p of the oracle confirms to the specification. However, even this approach does not solved all the problem as there is possibility of errors in specification. As a result, a device generated from the specification are correct, The result in the specification will correct and the result in specification will not be correct / reliable in the case of errors.

* Impractically Testing all data :→

For almost every program it is impossible to make an ~~any~~ attempt to test the program with all sets of input.

The correctness of the o/p for a particular test input is determined using testing oracle.

A testing oracle is a mechanism that can be used for determining whether a test as passed or failed.

* Impractically testing all paths :→

For almost every program it is impossible to attempt to test all executable paths through the project because of the no. of combinations that is combination explosion.

Developing an algo. for this purpose is also not possible.

5. a) What is boundary value analysis? Explain in detail with suitable example.

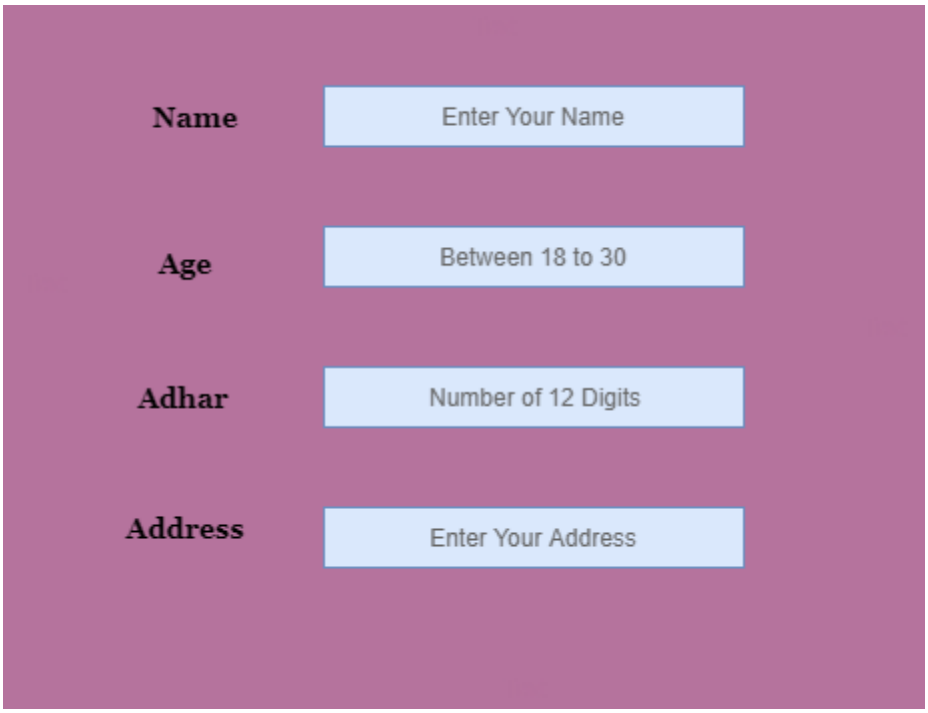
Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

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

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

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

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



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

Let's understand via practical:

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

for both valid and invalid partitions to capture the behavior of the system on different input conditions.

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

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

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

5. b) What is Equivalence class partitioning? Explain in detail with suitable example.

What is Equivalence Class Partitioning?

Equivalence class partitioning is a black-box testing technique or specification-based testing technique in which we group the input data into logical partitions called equivalence classes.
All the data items lying in an equivalence class are assumed to be processed in the same way by the software application to be tested when passed as input.

So, instead of testing all the combinations of input test data, we can pick and pass any of the test data from a particular equivalence class to the application and assume that the application will behave in the same way for the other test data of that class. Let's understand this with the help of an example.

Example

Consider an example of an application that accepts a numeric number as input with a value between 10 to 100 and finds its square. Now, using equivalence class testing, we can create the following equivalence classes-

Equivalence Class	Explanation
Numbers 10 to 100	This class will include test data for a positive scenario.
Numbers 0 to 9	This class will include test data that is restricted by the application. Since it is designed to work with numbers 10 to 100 only.
Greater than 100	This class will again include test data that is restricted by the application but this time to test the upper limit.
Negative numbers	Since negative numbers can be treated in a different way so, we will create a different class for negative numbers in order to check the robustness of the application.
Alphabets	This class will be used to test the robustness of the application with non-numeric characters.
Special characters	Just like the equivalence class for alphabets, we can have a separate equivalence class for special characters.

Identification of Equivalence Classes

- Cover all test data types for positive and negative test scenarios. We have to create test data classes in such a way that covers **all sets of test scenarios** but at the same time, there **should not be any kind of redundancy**.
- If there is a possibility that the test data in a particular class can be treated differently then it is better to **split that equivalence class**.

Advantages of Equivalence Classes Testing

- With the help of equivalence class testing, the number of test cases greatly reduces maintaining the same test coverage.
- This testing technique helps in delivering a quality product within a minimal time period.
- It is perfectly suitable for software projects with time and resource constraints.

Disadvantages of Equivalence Classes Testing

- The whole success of equivalence class testing relies on the identification of equivalence classes. The identification of these classes relies on the ability of the testers who creates these classes and the test cases based on them.
- In the case of complex applications, it is very difficult to identify all set of equivalence classes and requires a great deal of expertise from the tester's side.
- Incorrectly identified equivalence classes can lead to lesser test coverage and the possibility of defect leakage.

6. a) Differentiate between black box testing and white box testing.

S. No.	Black Box Testing	White Box Testing
1.	It 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.	It 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.
2.	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
3.	It is mostly done by software testers.	It is mostly done by software developers.
4.	No knowledge of implementation is needed.	Knowledge of implementation is required.
5.	It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
6.	It is a functional test of the software.	It is a structural test of the software.
7.	This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
8.	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
9.	It is the behavior testing of the software.	It is the logic testing of the software.
10.	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
11.	It is also called closed testing.	It is also called as clear box testing.
12.	It is least time consuming.	It is most time consuming.
13.	Example: Search something on google by using keywords	Example: By input to check and verify loops

6. b) What is state transition testing? Explain in detail with suitable example.

What is State Transition Testing?

State Transition Testing is a black box testing technique in which changes made in input conditions cause state changes or output changes in the Application under Test(AUT). State transition testing helps to analyze behaviour of an application for different input conditions. Testers can provide positive and negative input test values and record the system behavior.
It is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

State Transition Testing Technique is helpful where you need to **test different system transitions.**

When to Use State Transition?

- This can be used when a tester is testing the application for a finite set of input values.
- When the tester is trying to test sequence of events that occur in the application under test. I.e., this will allow the tester to test the application behavior for a sequence of input values.
- When the system under test has a dependency on the events/values in the past.

When to Not Rely On State Transition?

- When the testing is not done for sequential input combinations.
- If the testing is to be done for different functionalities like exploratory testing

Four Parts Of State Transition Diagram

There are 4 main components of the State Transition Model as below

1) States that the software might get

2) Transition from one state to another

3) Events that origin a transition like closing a file or withdrawing money

4) Actions that result from a transition (an error message or being given the cash.)

State Transition Diagram and State Transition Table

There are two main ways to represent or design state transition, State transition diagram, and state transition table.

In state transition diagram the states are shown in boxed texts, and the transition is represented by arrows. It is also called State Chart or Graph. It is useful in identifying valid transitions.

In state transition table all the states are listed on the left side, and the events are described on the top. Each cell in the table represents the state of the system after the event has occurred. It is also called State Table. It is useful in identifying invalid transitions.

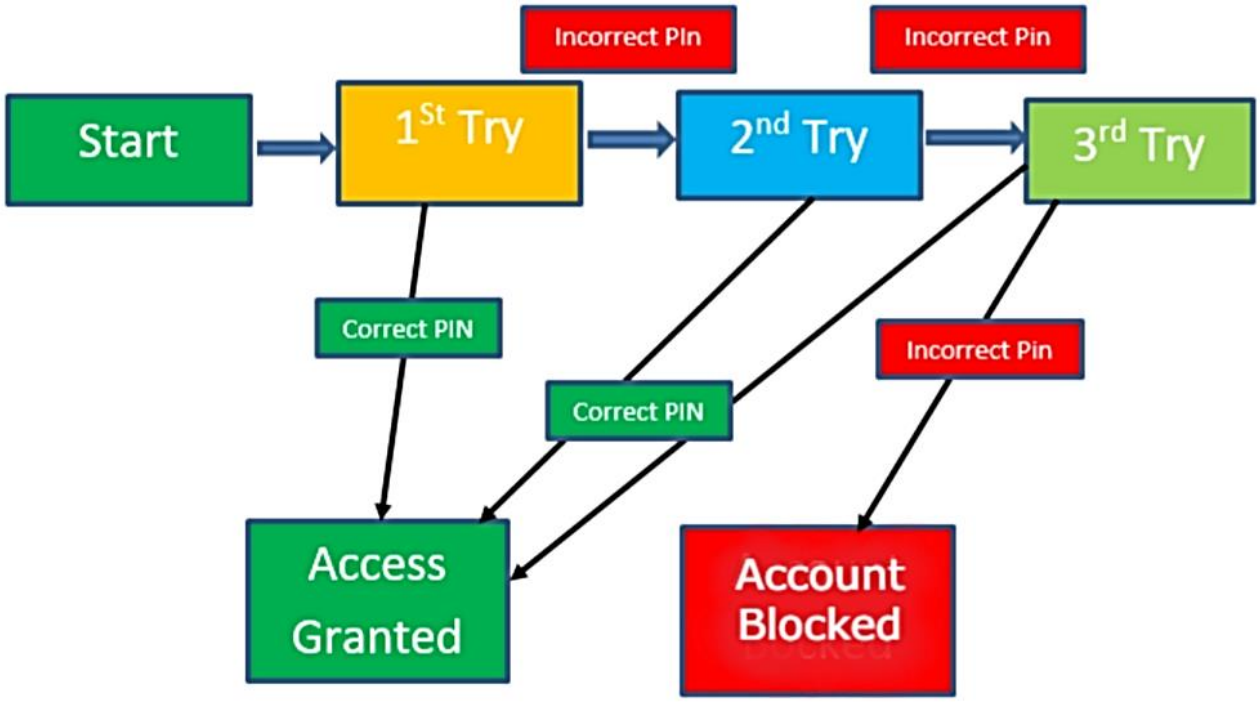
How to Make a State Transition (Examples of a State Transition)

Example 1:

Let's consider an ATM system function where if the user enters the invalid password three times the account will be locked.

In this system, if the user enters a valid password in any of the first three attempts the user will be logged in successfully. If the user enters the invalid password in the first or second try, the user will be asked to re-enter the password. And finally, if the user enters incorrect password 3rd time, the account will be blocked.

State transition diagram



In the diagram whenever the user enters the correct PIN he is moved to Access granted state, and if he enters the wrong password he is moved to next try and if he does the same for the 3rd time the account blocked state is reached.

State Transition Table

	Correct PIN	Incorrect PIN
S1) Start	**S5**	**S2**
S2) 1st attempt	**S5**	**S3**
S3) 2nd attempt	**S5**	**S4**
S4) 3rd attempt	**S5**	**S6**
S5) Access Granted	**–**	**–**
S6) Account blocked	**–**	**–**

In the table when the user enters the correct PIN, state is transitioned to S5 which is Access granted. And if the user enters a wrong password he is moved to next state. If he does the same 3rd time, he will reach the account blocked state.

7. a) Enlist any four tools to perform White box testing? Also explain advantages and disadvantages of white box testing.

White Box Testing Tools

Given below is a list of top white box test tools.

#1) Veracode

Veracode's white box testing tools will help you in identifying and resolving the software flaws quickly and easily at a reduced cost. It supports several application languages like .NET, C++, JAVA etc and also enables you to test the security of desktop, web as well as mobile applications. Still, there are several other benefits of Veracode tool. For detailed information about Veracode White box test tools, please check the below link.

Website Link: Veracode

#2) EclEmma

EclEmma was initially designed for test runs and analysis within the Eclipse workbench. It is considered to be a free Java code coverage tool and has several features as well. To install or know more about EclEmma please check out the below link.

Website Link: EclEmma

#3)RCUNIT

A framework which is used for testing C programs is known as RCUNIT. RCUNIT can be used accordingly based on the terms of the MIT License. It is free to use and in order to install or know more about it, please check the below link.

Website Link: RCUNIT

#4) cfix

cfix is one of the unit testing frameworks for C/C++ which solely aims at making test suites development as simple and easy as possible. Meanwhile, cfix is typically specialized for NT Kernel mode and Win32. To install and know more about cfix, please check out the below link

Website Link: cfix

#5) Googletest

Googletest is Google's C++ test framework. Test Discovery, Death tests, Value-parameterized tests, fatal & non-fatal failures, XML test report generation etc are few features of GoogleTest but there are several other features too. Linux, Windows, Symbian, Mac OS X are few platforms where GoogleTest has been used. In order to Download, please check the below link.

Download Link: Googletest

The various advantages and disadvantages of White Box Testing that are listed below:

Advantages of White Box Testing

The completeness, automation, time savings, optimization, and introspection of white box testing are its benefits.

1. Thoroughness

Complete code coverage is the fundamental tenet of white-box testing. The basic concept is to test as much code as you can, which is far more thorough than standard black-box testing. White-box testing's thoroughness also lends it a distinct framework. The rules of testing must be precise, engineering-based, and well-defined. This type of testing is transparent, it is possible to do thorough tests that cover all possible paths as well as the complete structure and code base. It evaluates internal and external vulnerabilities as well, which could aid in preventing future security threats and assaults.

2. Unit Testing

Unit testing is made possible by understanding the application's internal workings. As the name implies, unit tests examine single lines of code, or units, to determine whether they function as intended. These tests are easy to perform programmatically, allowing developers to rapidly determine whether something is broken. Unit tests are a useful tool for determining whether a previously functioning component has recently broken.

3. Time

Time management is a top responsibility during the software development process because there are constant deadlines to satisfy. White-box testing has the ability to drastically speed up the testing process. Developers frequently have a general understanding of the problem and the best way to resolve it as soon as they discover a fault. The expense of communication between developers and QA is also eliminated by white-box testing because developers can identify and address problems without waiting for QA.

4. Optimization

A section-by-section analysis of the code enables developers to eliminate unnecessary code or condense already-existing code. Additionally, by removing obfuscated problems that can go undetected during routine testing, code can be made more efficient.

5. Introspection

White-box testing enables programmers to thoroughly consider implementation. Developers are compelled to think about the relationships between different pieces of code. Perhaps the existing implementation is adequate but will not scale well in the future or contains extraneous components that can be removed. Developers can review designs and consider how they might be improved by using white-box testing.

Disadvantages of White Box Testing

White-box testing has drawbacks including high cost, frequently changing code, and missing cases.

1. Expensive

White-box testing becomes highly time and money-consuming to undertake as it is more thorough. Although this is somewhat mitigated by unit testing, writing the unit tests requires an initial investment. Additionally, this kind of testing may not scale well with huge applications. Testing every code version becomes very difficult. White-box testing necessitates competent testers who are familiar with programming, in contrast to black-box testing. This drives up the cost and may discourage developers from working on additional features. White-box testing must take all of these costs into account.

2. Code base that changes quickly

If the code base is changing quickly, automated test cases are useless. Most written test cases are frequently useless after redesigns or rework and require rewriting. If the implementation changes frequently, an updated test script is necessary.

3. Incomplete cases

Only existing functionalities are validated and tested during white-box testing. White-box testing won't detect a feature that is only partially implemented or that has certain components missing. Black-box testing that is driven by requirements excels in this area.

4. Time consuming

When employing the white box testing approach for large applications, exhaustive testing becomes even more difficult. White box testing takes a lot of time because it requires creating a wide range of inputs to test every possible path and circumstance.

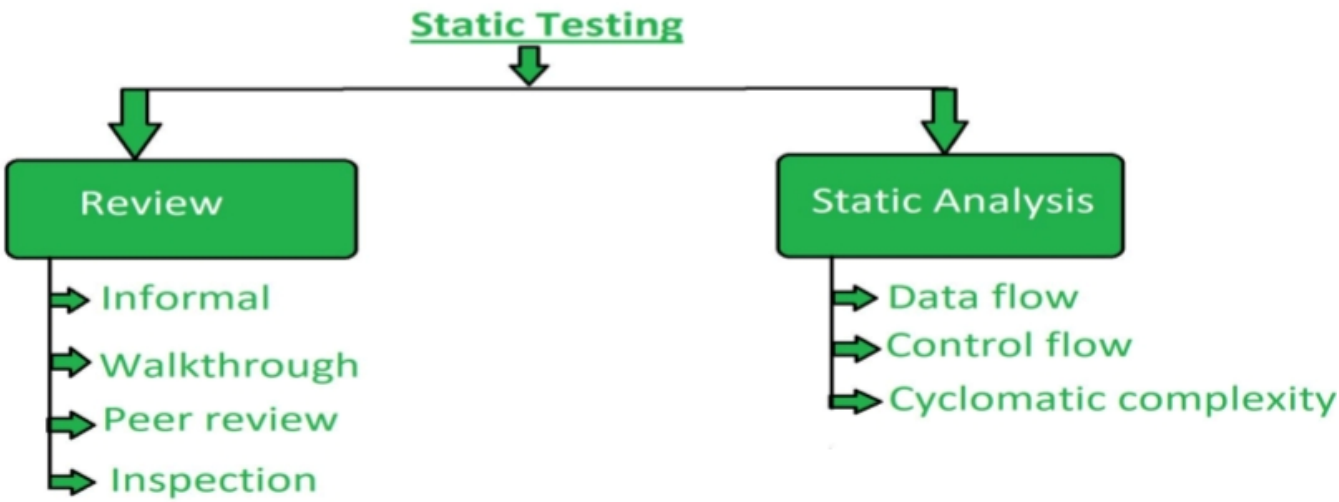
5. More errors

It is not realistic to test every condition, thus some might go untested. With the general method of analysing each line by line or path by path, errors in the code might not be found or might even be introduced.

7. b) What is static testing? Explain in detail with suitable example.

Static Testing is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application. Whereas in Dynamic Testing checks, the code is executed to detect the defects. Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that cannot be found using Dynamic Testing, can be easily found by Static Testing. **Static Testing**

Techniques: There are mainly two type techniques used in Static Testing:



Review: In static testing review is a process or technique that is performed to find the potential defects in the design of the software. It is process to detect and remove

errors and defects in the different supporting documents like software requirements specifications. People examine the documents and sorted out errors, redundancies and ambiguities. Review is of four types:

- **Informal:** In informal review the creator of the documents put the contents in front of audience and everyone gives their opinion and thus defects are identified in the early stage.
- **Walkthrough:** It is basically performed by experienced person or expert to check the defects so that there might not be problem further in the development or testing phase.
- **Peer review:** Peer review means checking documents of one-another to detect and fix the defects. It is basically done in a team of colleagues.
- **Inspection:** Inspection is basically the verification of document the higher authority like the verification of software requirement specifications (SRS).

2. **Static Analysis:** Static Analysis includes the evaluation of the code quality that is written by developers. Different tools are used to do the analysis of the code and comparison of the same with the standard. It also helps in following identification of following defects:

```
(a) Unused variables
(b) Dead code
(c) Infinite loops
(d) Variable with undefined value
(e) Wrong syntax
```

Static Analysis is of three types:

- **Data Flow:** Data flow is related to the stream processing.
- **Control Flow:** Control flow is basically how the statements or instructions are executed.
- **Cyclomatic Complexity:** Cyclomatic complexity defines the number of independent paths in the control flow graph made from the code or flowchart so that minimum number of test cases can be designed for each independent path.

8. a) What is McCabe's Cyclomatic Complexity? Compute Cyclomatic Complexity of following program code.

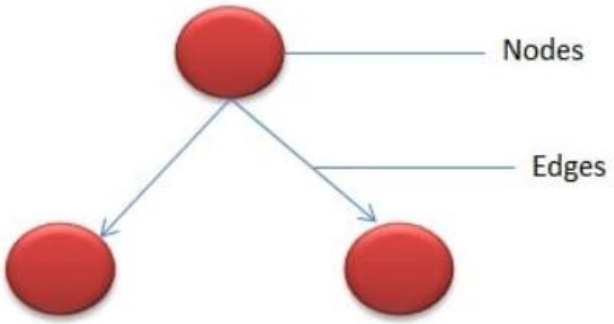
```
i = 0;
n=4; //N-Number of nodes present in the graph
while (i<n-1) do
    j = i + 1;
    while (j<n) do
        if A[i]<A[j] then
        swap(A[i], A[j]);
    end do;
    j=j+1;
end do;
```

What is McCabe's Cyclomatic Complexity?

Cyclomatic Complexity in Software Testing is a testing metric used for measuring the complexity of a software program. It is a quantitative measure of independent paths in the source code of a software program. Cyclomatic complexity can be calculated by using control flow graphs or with respect to functions, modules, methods or classes within a software program.
Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths.

This metric was developed by Thomas J. McCabe in 1976 and it is based on a control flow representation of the program. Control flow depicts a program as a graph which consists of Nodes and Edges.

In the graph, Nodes represent processing tasks while edges represent control flow between the nodes.

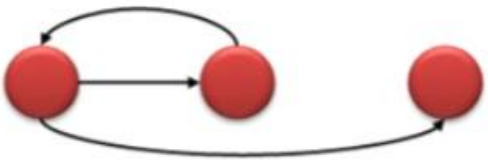


Flow graph notation for a program:

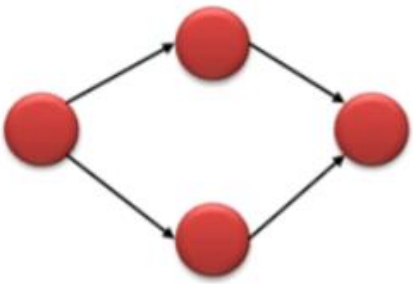
Flow Graph notation for a program defines several nodes connected through the edges. Below are Flow diagrams for statements like if-else, While, until and normal sequence of flow.

Sequence

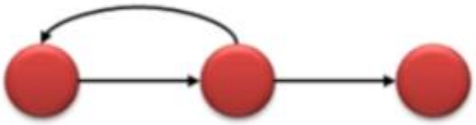
While



If-then-else



Until



How to Calculate Cyclomatic Complexity

Mathematical representation:

Mathematically, it is set of independent paths through the graph diagram. The Code complexity of the program can be defined using the formula –

```
V(G) = E - N + 2
```

Where,

E – Number of edges

N – Number of Nodes

```
V (G) = P + 1
```

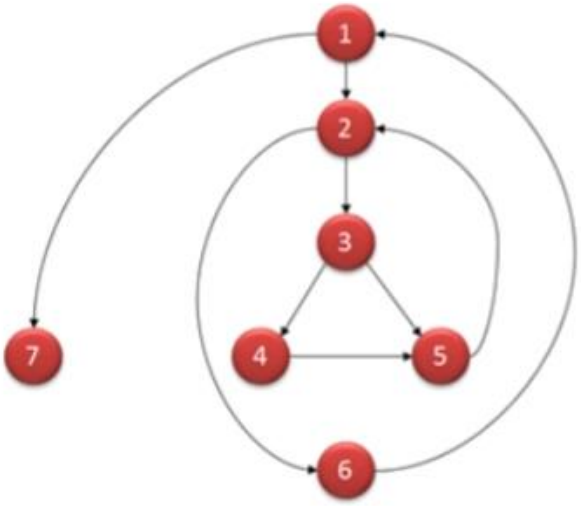
Where P = Number of predicate nodes (node that contains condition)

Example –

```
i = 0;
n=4; //N-Number of nodes present in the graph
```

```
while (i<n-1) do
j = i + 1;

while (j<n) do

if A[i]<A[j] then
swap(A[i], A[j]);

end do;
j=j+1;

end do;
```

Flow graph for this program will be



Computing mathematically,

- V(G) = 9 – 7 + 2 = 4
- V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)
- Basis Set – A set of possible execution path of a program
- 1, 7
- 1, 2, 6, 1, 7
- 1, 2, 3, 4, 5, 2, 6, 1, 7
- 1, 2, 3, 5, 2, 6, 1, 7
- ollowing table gives overview on the complexity number and corresponding meaning of v (G):

Complexity Number	Meaning
1-10	Structured and well written code High Testability

	Cost and Effort is less
10-20	Complex Code Medium Testability Cost and effort is Medium
20-40	Very complex Code Low Testability Cost and Effort are high
>40	Not at all testable Very high Cost and Effort

8. b) Enlist any two tools for computation of Cyclomatic Complexity? How does Cyclomatic complexity is useful in software testing?

TOOLS: Cyclo, CCCC, PMD, Findbugs, LC2

Cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it. It is a software metric used to indicate the complexity of a program. It is computed using the Control Flow Graph of the program. The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes

Use of Cyclomatic Complexity:

- Determining the independent path executions thus proven to be very helpful for Developers and Testers.
- It can make sure that every path have been tested at least once.
- Thus help to focus more on uncovered paths.
- Code coverage can be improved.
- Risk associated with program can be evaluated.
- These metrics being used earlier in the program helps in reducing the risks.

Cyclomatic complexity in software testing is used to measure the number of logical paths of a program. Basically, it evaluates the complexity of linearly independent paths in the source code of a program. For example, if the number of paths/points is more, the program will be more complex.

9. a) What is unit testing? Explain in detail with suitable example.

What is Unit Testing?

Unit testing is testing the smallest testable unit of an application.

It is done during the coding phase by the developers.

To perform unit testing, a developer writes a piece of code (unit tests) to verify the code to be tested (unit) is correct.

A Real-world Example

You have written a function to add two numbers:

```
int Add(int a, int b) { return a+b; }
```

The above function takes two numbers as input and returns their sum.

A unit test code would look something like this:

```
void TestAdd1() { Assert.IsEqual(Add(5, 10), 15) }
```

The above unit test "asserts" that 5 + 10 is equal to 15. If the Add function returns anything else Assert.IsEqual result in error and the test case will fail.

You will probably add a few more unit test cases like these:

```
void TestAdd2() { Assert.IsEqual(Add(500, 1000), 1500) }
void TestAdd3() { Assert.IsEqual(Add(0, 1000), 1000) }
void TestAdd4() { Assert.IsEqual(Add(-100, 100), 0) }
void TestAdd5() { Assert.IsEqual(Add(-100, -1100), -1200) }
```

After you write your test cases, you will run them to verify that everything is working correctly.

Later another developer, in addition to adding some of his code, accidentally modifies the **Add** function as:

```
int Add(int a, int b) { return a*b; }
```

As you can see, instead of adding the two numbers, the code now multiplies them.

9. b) Explain in detail about advantages and disadvantages of unit testing.

Disadvantages of Unit Testing

Unit tests can be a lousy investment if not done correctly.

Unit testing increases the initial development time.
It takes time to write good test cases. Since a developer does this, it adds to the development time. In the long run, good test cases can save time, but many think of unit testing as an expense rather than an investment and do not allocate the required amount of time.
Unit testing can lead to an unnecessary proliferation of test cases leading to increased maintenance time.

We have often seen a hundred test cases when just twenty would have sufficed. This inflation usually happens when someone other than the developer writes the test cases, and the incentive is based on the number of test cases written rather than the quality of the test cases. When the underlying API changes, reworking the test cases takes time but it never added to the code's quality in the first place.
Unit testing can induce false confidence in the quality of the code.
It is vital to ensure that unit tests examined all the code paths. To do this, developers have to install a code coverage software and verify the unit tests' efficacy. But this is rare. Unit testing is more often than not considered a to-do item rather than a necessity. As a result, you may often see a lot of redundant test cases and a lot of necessary boundary condition testing missing. Build

Disadvantages of Unit Testing:

1. The process is time-consuming for writing the unit test cases.
2. Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
3. Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.
4. It requires more time for maintenance when the source code is changed frequently.
5. It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.
6. Time and Effort: Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.
7. Dependence on Developers: The success of unit testing depends on the developers, who must write clear, concise, and comprehensive test cases to validate the code.
8. Difficulty in Testing Complex Units: Unit testing can be challenging when dealing with complex units, as it can be difficult to isolate and test individual units in isolation from the rest of the system.
9. Difficulty in Testing Interactions: Unit testing may not be sufficient for testing interactions between units, as it only focuses on individual units.
10. Difficulty in Testing User Interfaces: Unit testing may not be suitable for testing user interfaces, as it typically focuses on the functionality of individual units.
11. Over-reliance on Automation: Over-reliance on automated unit tests can lead to a false sense of security, as automated tests may not uncover all possible issues or bugs.
12. Maintenance Overhead: Unit testing requires ongoing maintenance and updates, as the code and test cases must be kept up-to-date with changes to the software.

Advantages of Unit Testing:

1. Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
2. Unit testing allows the programmer to refine code and make sure the module works properly.
3. Unit testing enables testing parts of the project without waiting for others to be completed.
4. Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process, before they become larger and more difficult to fix.
5. Improved Code Quality: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.
6. Increased Confidence: Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.
7. Faster Development: Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.
8. Better Documentation: Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.
9. Facilitation of Refactoring: Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.
10. Reduced Time and Cost: Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.

10. a) What is integration testing? Explain in detail with suitable example.

What is integration testing?

Integration testing -- also known as integration and testing (I&T) -- is a type of software testing in which the different units, modules or components of a software application are tested as a combined entity. However, these modules may be coded by different programmers.

The aim of integration testing is to test the interfaces between the modules and expose any defects that may arise when these components are integrated and need to interact with each other.

Common approaches to integration testing

Four key strategies to execute integration testing are big-bang, top-down, bottom-up and sandwich/hybrid testing. Each approach has benefits and drawbacks.

Big-bang testing: The big-bang approach involves integrating all modules at once and testing them all as one unit.

Big-bang testing's advantages include the following:

- Its suitability for testing small systems.
- Its ease of identifying errors in such systems, saving time and speeding up application deployment.

However, big-bang testing has disadvantages, for example:

- Locating the source of defects can be difficult since different modules are integrated as one unit.
- Big-bang testing is time-consuming for a large system with numerous units.
- Testers could miss some interface links or bugs.
- Testers must wait until all modules are available, so they have less time to do the testing and developers have less time to fix any errors.
- Due to simultaneous testing, high-risk critical modules and peripheral modules dealing with user interfaces are not tested on priority (as they should be).

Top-down testing: The top-down approach is an incremental approach that involves testing from the topmost or highest-level module and gradually proceeding to the lower modules. Each module is tested one by one, and then integrated to check the final software's functionality.

Advantages of top-down testing are as follows:

- It is easier to identify defects and isolate their sources.
- Testers check important units first, so they are more likely to find critical design flaws.

- Testers must wait until all modules are available, so they have less time to do the testing and developers have less time to fix any errors.
- Due to simultaneous testing, high-risk critical modules and peripheral modules dealing with user interfaces are not tested on priority (as they should be).

Top-down testing: The top-down approach is an incremental approach that involves testing from the topmost or highest-level module and gradually proceeding to the lower modules. Each module is tested one by one, and then integrated to check the final software's functionality.

Advantages of top-down testing are as follows:

- It is easier to identify defects and isolate their sources.
- Testers check important units first, so they are more likely to find critical design flaws.

- It is possible to create an early prototype.

However, disadvantages to top-down testing are as follows:

- The examination of lower-level modules can take a lot of time, so testers may not test them adequately or completely.
- When too many testing stubs are involved, the testing process can become complicated.

Bottom-up testing: Bottom-up (also known as bottom-to-top) integration testing is the opposite of the top-down approach. It involves testing lower-level modules first, and then gradually progressing incrementally to higher-level modules. This approach is suitable when all units are available for testing.

Advantages of bottom-up testing are as follows:

- It is easier to find and localize faults.
- Less time is needed for troubleshooting since testers don't have to wait for all modules to be available for testing.

Meanwhile, disadvantages of this type of testing include the following:

- Testing all modules can take a lot of time, so there may be delays in releasing the final product.
- Critical modules are tested only in the final stages, so testers may miss some defects and developers may not have enough time to fix found defects.
- Testing can be complicated if the software consists of multiple low-level units.
- It is not possible to create an early prototype.

What are the key steps in integration testing?

Integration testing example

Consider a video-streaming mobile application. Its core features include the following:

- Sign up/log in.
- View different monthly/yearly subscription plans.
- Choose personalized plans.
- Watch streaming video.

Once users download the applications, they see a sign-up form where they can enter their account information. After successful authorization, they are redirected to a page listing different subscription plans. They can choose their own plan and then complete the payment.

Any errors in this logical flow could cause problems for the user and lead to losses for the app company. Integration testing can help find and fix such errors.

So, after each module is ready, testers conduct unit testing. And once all modules are available, testers test them together to check their interfaces and data flows. If no errors are detected, the end user should be able to successfully complete their transaction.

10. b) Explain in detail about advantages and disadvantages of integration testing.

Advantages of integration testing:

- Integration testing provides a systematic technique for assembling a software system while conducting tests to uncover errors associated with interfacing.
- The application is tested in order to verify that it meets the standards set by the client as well as reassuring the development team that assumptions which were made during unit testing are correct.
- Integration testing need not wait until all the modules of a system are coded and unit tested. Instead, it can begin as soon as the relevant modules are available.
- Integration testing or incremental testing is necessary to verify whether the software modules work in unity.
- System Integration testing includes a number of techniques like Incremental, Top- down, Bottom –Up, Sandwich and Big Bang Integration techniques.

Disadvantages of Integration testing :

- Fault Localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some interfaces links to be tested could be missed easily.
- Since the integration testing can commence only after "all" the modules are designed, testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.
- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- Early prototype is not possible
- Needs many Stubs.