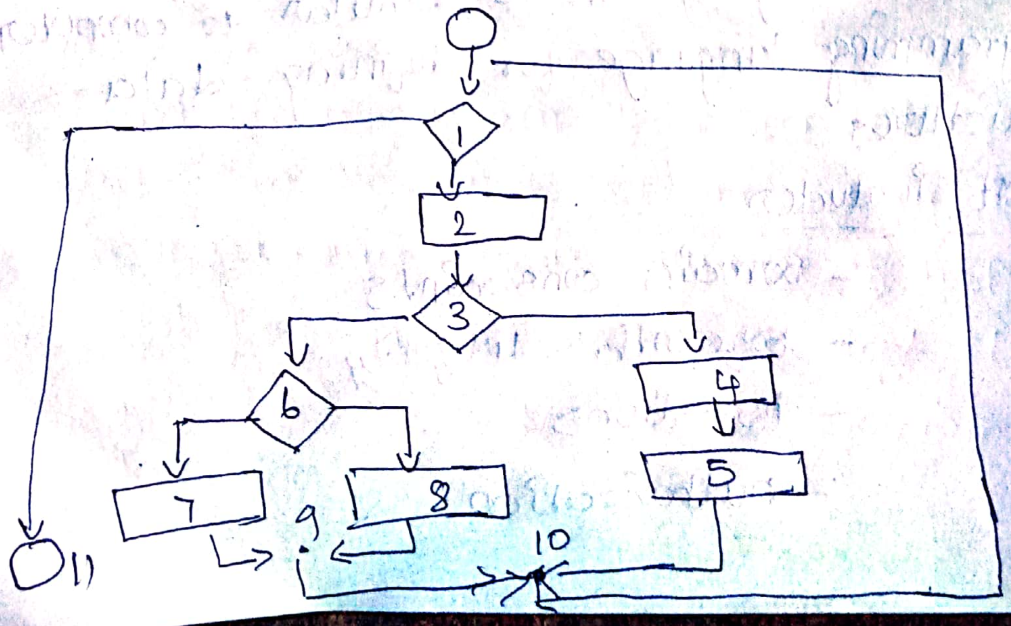
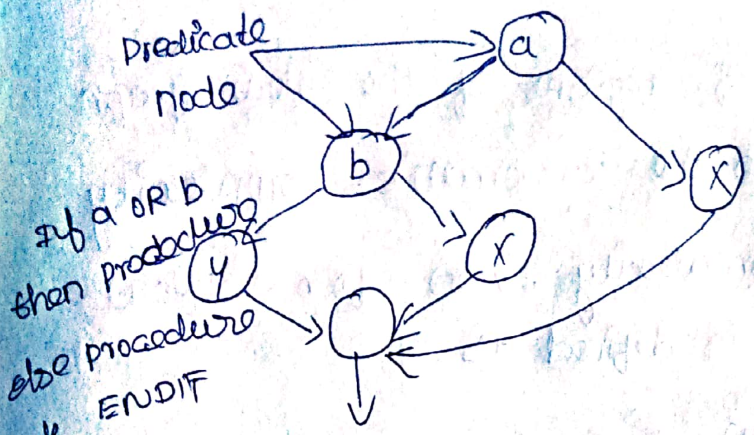
1) Explain in detail cyclomatic complexity and its calculation with example.

Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program. when used in the context of the basic path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

An independent path is any path through the program that introduces at least one new set of processing statements or a new condition. when stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.





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

path 1, 2, 3 and 4 constitute a basis set for the flow graph in figure 4.3B. That is if tests can be designed to force execution of these paths, every statement in the program will have been guaranteed to be executed at least one time and every condition will have been executed on its true and false sides. It should be noted that the basis set is not unique. In fact, a number of different basis sets can be derived for a given procedural design.

cyclomatic complexity has a foundation in graph theory and provides us with an extremely useful software metric, complexity is computed in one of three ways.

1. The number of regions of the flow graph correspond to the cyclomatic complexity

2. cyclomatic complexity, V(G), for a flow graph G, is defined as.

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

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.

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. More important, the value for V(G) provides us with an upper bound for the number of independent path that from the basis set and by implication an upper bound on the number of tests that must be designed and executed to guarantee coverage of all program statements.

2) Explain the Regression Testing and Integration Testing in detail.

Regression Testing:

Each time a new module is added as part of integration testing, the software changes new data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly.

In the context of an integration test strategy, regression testing is the reexecution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

In a broader context, successful tests result in the discovery of errors. and errors must be corrected. whenever software is corrected, some aspect of the software configuration is changed. Regression testing is the activity that helps to ensure that changes do not introduce unintended behavior or additional errors.

=> A representative sample of tests that will exercise all software functions.

=> Additional tests that focus on software functions that are likely to be affected by the change.

=> Tests that focus on the software components that have been changed.

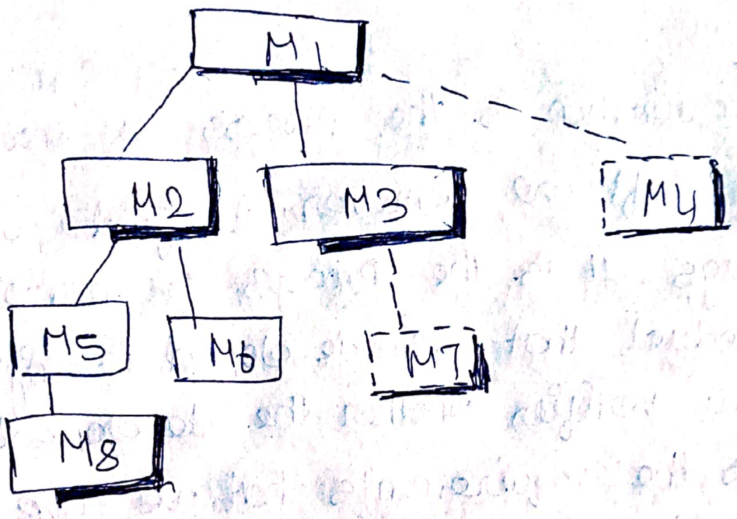
As integration testing proceeds, the number of regression tests can grow quite large.

Integration Testing:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design.

Top-down integration:

Top-down integration testing is an incremental approach to construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module. Modules subordinate to the main control module are incorporated into the structure in either a depth first or breadth -first manner.



Bottom-up Integration:

Bottom-up integration testing, as its name implies, begins construction and testing with atomatic modules. Because components are integrated from the bottom-up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

A boottom-up integration statogy may be implemented with the following steps.

=> A driver is written to coordinated test case input and output

=> The cluster is tested.

3) Differentiate verification and validation. which type of testing address verification? which type of testing address validation?

Verification :

verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not It verifies whether the developed product fulfills the requirements that we have. verification is static testing.

Validation :

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product It is validation of actual and expected product. validation is the dynamic testing.

Four Types of verification:

⇒ component Testing, verifying a software element.

=> Integration Testing, verifying if the units work together.

=> system Testing, verifying that the system meets the requirements

=> Acceptance Testing, verifying that the system satisfies accepance criteria.

and the four types are:

- Inspection
- Analysis
- Testing
- Demonstration.

Four types of validation:

According to Tutorialspoint, validation testing in the V model has the four activities:

=> unit Testing, validating the program.

=> Integration Testing, validating the design

=> System testing, validating the system / architecture

=> User Acceptance Testing, validating against requirements.

4) what is black box testing? Explain the different type of black box testing stratogic Explain by considering suitable examples.

Black Box Testing:

Black box testing is also known as Functional testing or Behavioural testing. It is an effective and efficient approach used to concentrate on the inputs, outputs and principle function of a software system module. In a block-box testing, the test of a software is based on system specifications rather than on code Thus, the system is assumed to be 'black box' whose behaviour can only be determined by studying its inputs, outputs and functional requirements of the software. The tests can be observerd from the program codes codes or component specification.

Black box testing is helpful to software engineers in order to understand the set of input conditions, which define overall functional requirements of a program. Black-box testing is useful to identify errors such as.

(i) Inaccurate functions

(ii) Interface errors

(iii) performance errors

(iv) · Data structure errors

(v) initialization and termination errors.

Different Types of Black Box Testing:

(i) **Equivalence class partitioning:**

Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from test cases can be derived.

Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition. using concepts, if a set of objects can be linked by relationships that are symmetric, transitive, and reflexive, an equivalence class is present.

Boundary value Analysis:

A greater number of errors tend to occur at the boundaries of the input domain rather than in the "center". It is for this reason that boundary value analysis has been developed as a testing technique.

Boundary value analysis is a test case design technique that complements equivalental partitioning.

(iii) Orthogonal Testing :

Orthogonal testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing. The orthogonal array testing method is particularly useful in finding errors associated with region faults - an error category associated with faulty logic within a software component

(iv) Graph Based Testing method :

The first step in black-box testing is to understand the objects that are modeled in software and the relationships that connect those objects. Once this has been accomplished, the next step is to define a series of tests of that verify "all objects have the expected relationship to one another".

Advantages :

=> Test cases are performed from the user's perspective.

=> It reveals the problems or anomalies in the specifications

5) Explain in detail about white box testing.

White - Box testing:

white Box testing deals with the internal logic and structure of the program code. It is also known as glass-box, structural or open-box testing. In glass-box testing, test classes are derived based on the knowledge of software structure and its implementation.

(i) statement coverage
(ii) Branch/Decision coverage.
(iii) condition coverage.

(i) statement coverage:

Test values are provided to check whether each statement in the module is executed at least once. statement testing executes statements when,

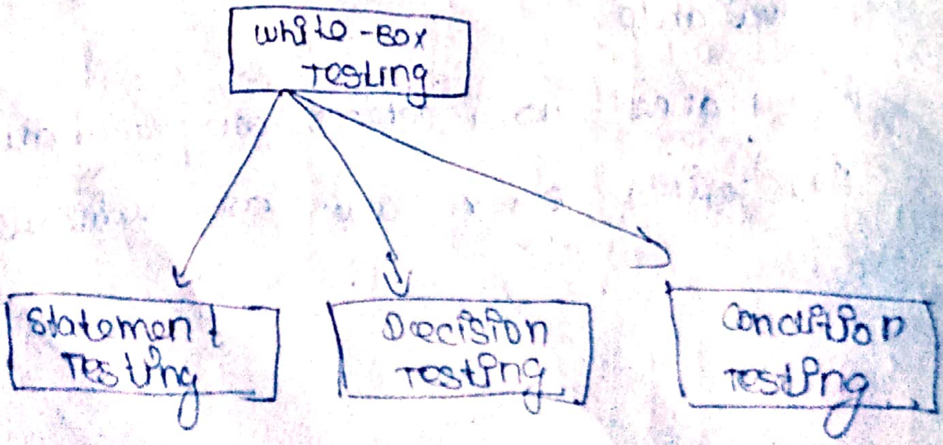
(i) optional arguments are available
(ii) user-provided parameters or procedures are available.
(iii) planned user-actions are available
(iv) defined error codes are available.

(ii) Branch / Decision coverage

Tests are performed to check whether each branch of a decision is executed atleast once. Decisions require two values in standard Boolean decision testing. But in case of component or nested decisions, the number of Boolean decision values can be greater than two.

(iii) condition coverage:

Tests are performed to check whether each condition in a decision accepts all the necessary outputs at least once. It also checks whether the entry points to the procedure or flow is invoked at least once. In case of compound and nested loops, condition testing is provided with multiple test values. The other exceptional routines and error handlers are also invoked while testing the entry points.



b) write elaborately on unit testing. How do you develop test case.

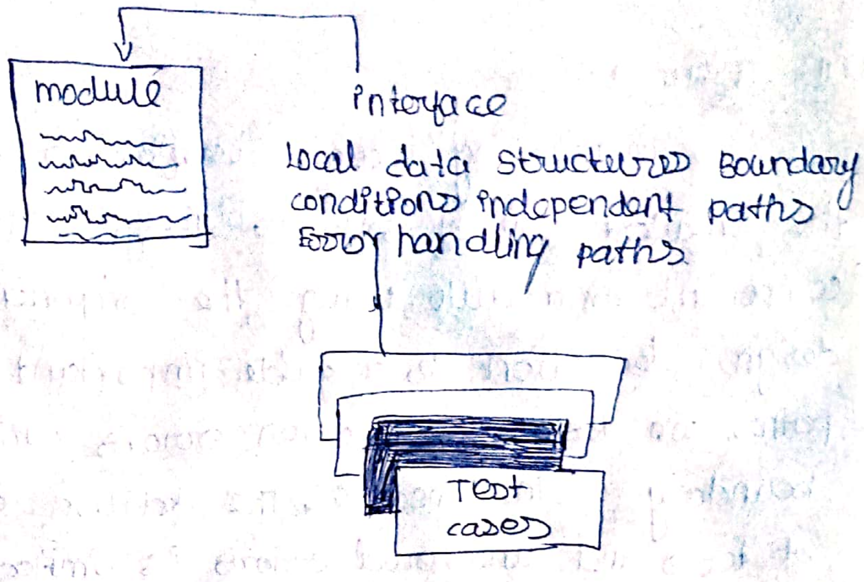
UNIT TESTING:

UNIT testing focuses verification effort on the smallest unit of software design - the software component or module. Using the component-level design description as a guide important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components.

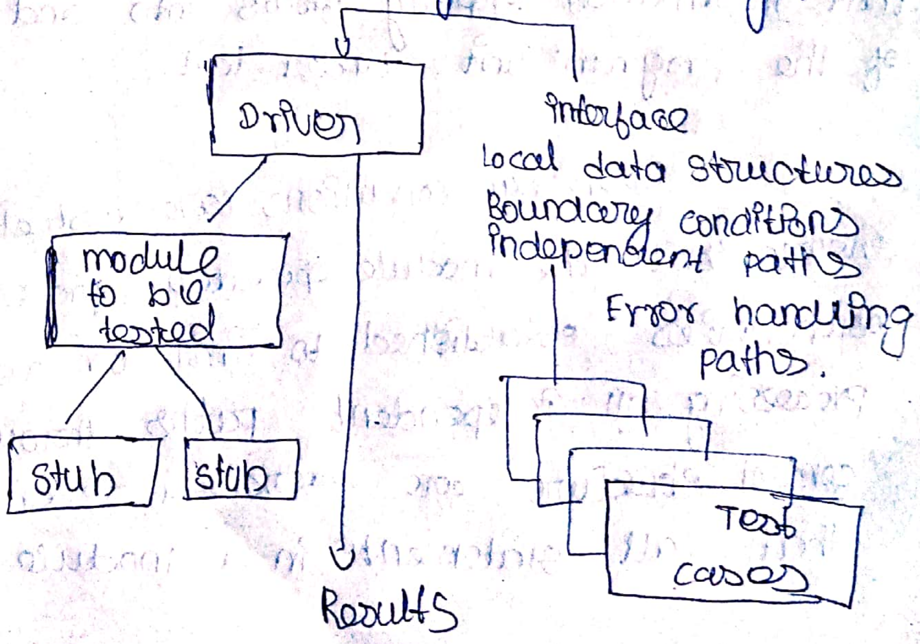
Unit Test considerations:

The module interface is tested to ensure that information properly flows into and out of the program unit under test.

Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths are tested.


module
interface
local data structures Boundary conditions independent paths Error handling paths
Test cases

Tests of data flow across a module interface are required before any other test is initiated. If data do not enter and exit propely, all other tests are moot. In addition, local data structures should be exercised and the local impact on glocal data should be ascertained during unit testing.



Scanned with CamScanner

Unit Test Procedures:

After source level code has been developed, reviewed, and verified for correspondence to component guidance for establishing test cases Each test case should be coupled with a set of expected results. Because a component is not a stand alone program, driver and (or stub software must be developed for each unit test. The unit test environment is illustrated in Figure.

Drivers and stubs represent overhead. That is, both are software that must be written but that is not delivered with the final software product. If drivers and stubs are kept simple, actual overhead is relatively low. unit testing is simplified when a component with high cohesion is designed.

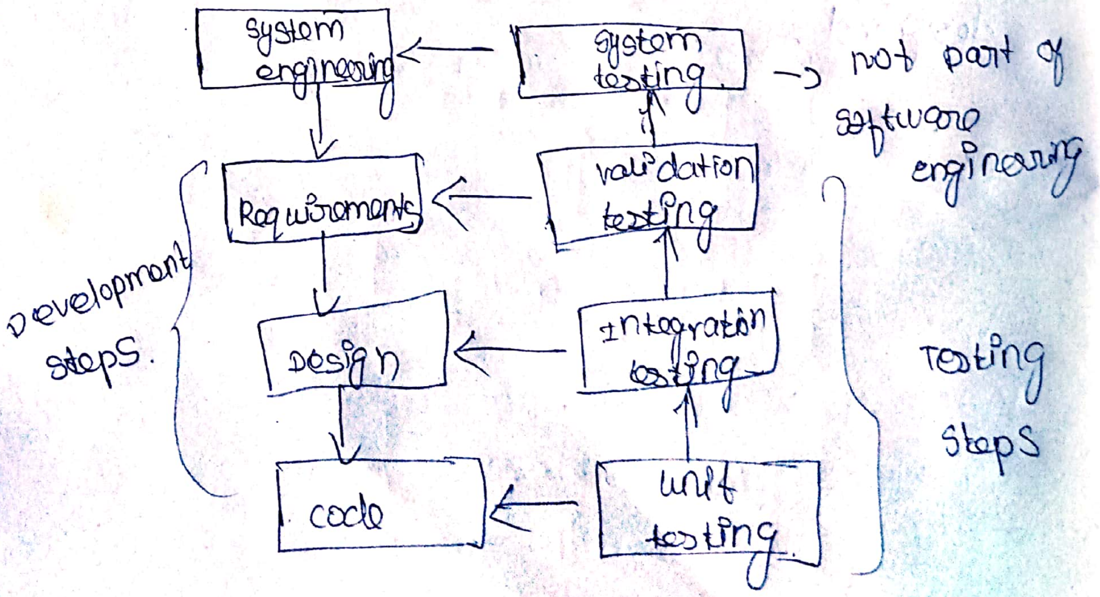
Scanned with CamScanner

7 Explain following tests :
i) system testing
ii) performance Testing
iii) Acceptance Testing

i) system Testing:

with the end of software validation, the role the software engineer also ends. The system engineer combines the validated software with other system components like database, and hardware, and performs systems testing. The testing ensures that all system components work together and performs according to the system requirements.

The testing and development steps are depicted in the below figure.



(ii) performance Testing

performance Testing is a Type of software testing that ensures software application to perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity and stability under a particular workload.

performance Testing is the process of analyzing the quality and capability of a product. It is a testing method performance in terms of speed, reliability and stability under varying workload. performance testing is also known as perf Testing.

performance Testing Attributes:

=> speed

=> scalability

=> stability

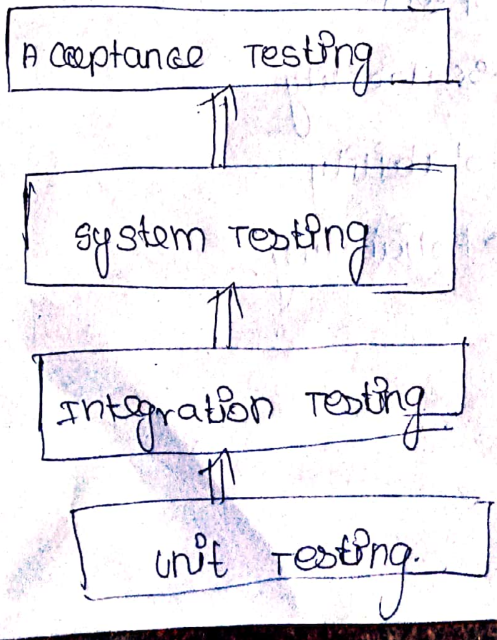
=> Reliability.

(iii) Acceptance Testing:

Acceptance Testing is a method of software testing where a system is tested for acceptability. The major aim of this test is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not.

Standard Definition of Acceptance Testing:

It is a formal testing according to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users customers or other authorized entities to determine whether to accept the system or not.


Acceptance Testing
System Testing
Integration Testing
Unit Testing.