SEPM QUESTION BANK SOLUTION

What do you mean by data modeling? Explain.

A data model shows the client's information needs and business processes through entities, relationships and data required within the system. It complements the data flow diagram which shows how the data is processed. Data models can be conceptual (high level entities and relationships to document business concepts or high level requirements), logical (more detailed information on entities, attributes and relationships by often expanding the conceptual model to include attributes, columns, fields and keys) or physical (how data is stored and managed in an application).
Data models are diagrams supported by textual descriptions. They can include people, places, things, concepts, attributes and relationships. Textual descriptions are usually included in a data dictionary.

Data modeling is the process of creating a simplified diagram of a software system and the data elements it contains, using text and symbols to represent the data and how it flows. Data models provide a blueprint for designing a new database or reengineering a legacy application. Overall, data modeling helps an organization use its data effectively to meet business needs for information.

A data model can be thought of as a flowchart that illustrates data entities, their attributes and the relationships between entities. It enables data management and analytics teams to document data requirements for applications and identify errors in development plans before any code is written.

Alternatively, data models can be created through reverse-engineering efforts that extract them from existing systems. That's done to document the structure of relational databases that were built on an ad hoc basis without upfront data modeling and to define schemas for sets of raw data stored in data lakes or NoSQL databases to support specific analytics applications.

Define following software design concepts :

Abstraction

Pattern

Modularity

Information Hiding

Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important. If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we are able to divide the system into components then the cost would be small.

The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.

Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can’t be accessed by any other modules.

A. Give & explain ten design principles in detail.

- The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

4

How to achieve the Quality

- A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics and (3) can be implemented in an evolutionary fashion
  - For smaller systems, design can sometimes be developed linearly.
- A design should be modular; that is, the software should be logically partitioned into elements or subsystems
- A design should contain distinct representations of data, architecture, interfaces, and components.
- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communicates its meaning.

5

B. Define following software design concepts :

a. Refactoring

b. Class & object

c. Encapsulation

Refactoring is **the process of changing a software system in such a way that it does not alter the function of the code yet improves its internal structure**.

A class is a blueprint from which you can create the instance, i.e., objects. An object is the instance of the class, which helps programmers to use variables and methods from inside the class. A class is used to bind data as well as methods together as a single unit. Object acts like a variable of the class.

Encapsulation is **a way to restrict the direct access to some components of an object**, Encapsulation can be used to hide both data members and data functions or methods associated with an instantiated class or object.

What is modularity? How to find moderate number of modules required with moderate cost of software?

The module simply means the software components that are been created by dividing the software. The software is divided into various components that work together to form a single functioning item but sometimes they can perform as a complete function if not connected with each other. This process of creating software modules is known as **Modularity** in software engineering.

Explain data flow diagram in detail. Give the extension suggested by ward and mellor.

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The following observations about DFDs are essentialatch

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Levels of DFD

DFD uses hierarchy to maintain transparency thus multilevel DFD's can be created. Levels of DFD are as follows:

- 0-level DFD
- 1-level DFD:
- 2-level DFD:

Ward and Mellor extend basic structured analysis notation to accommodate the following demands imposed by a real-time system:

- Information flow is gathered or produced on a time-continuous basis.
- Control information is passed throughout the system and associated control processing.
- Multiple instances of the same transformation are sometimes encountered in multitasking situations.
- Systems have states and a mechanism causes transition between states.

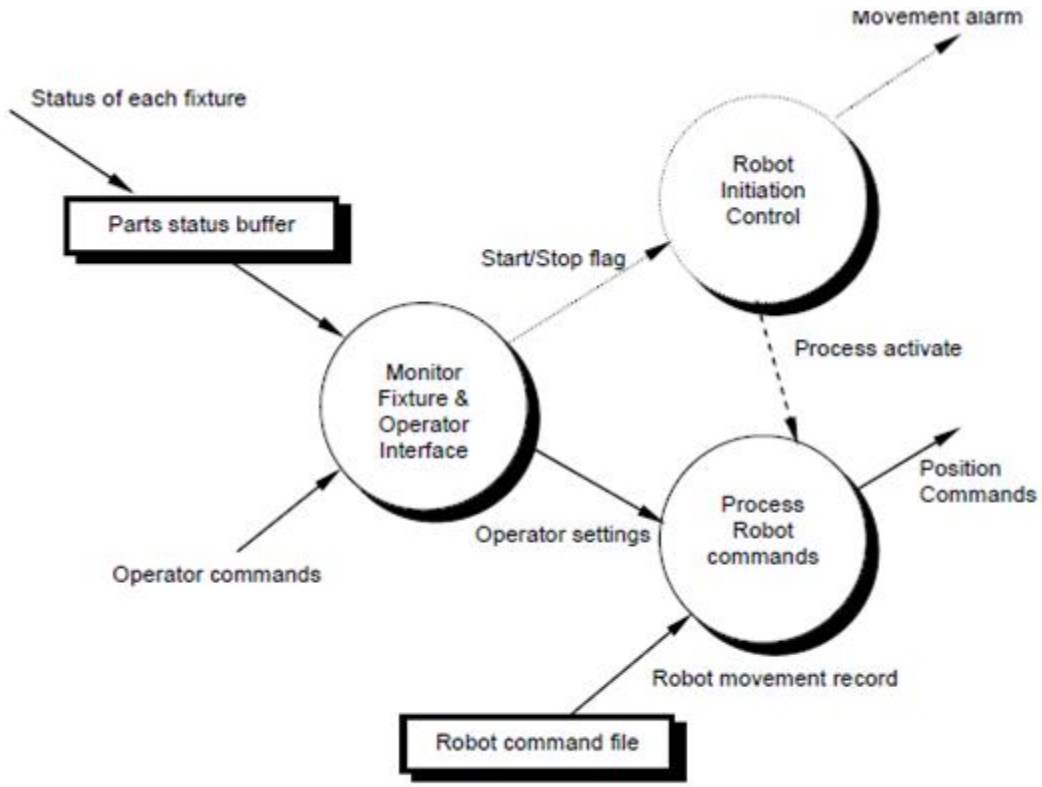


Figure . . . Data and control flows using ward and mellor notation

What is Cyclomatic Complexity?Explain how it is computed?

Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

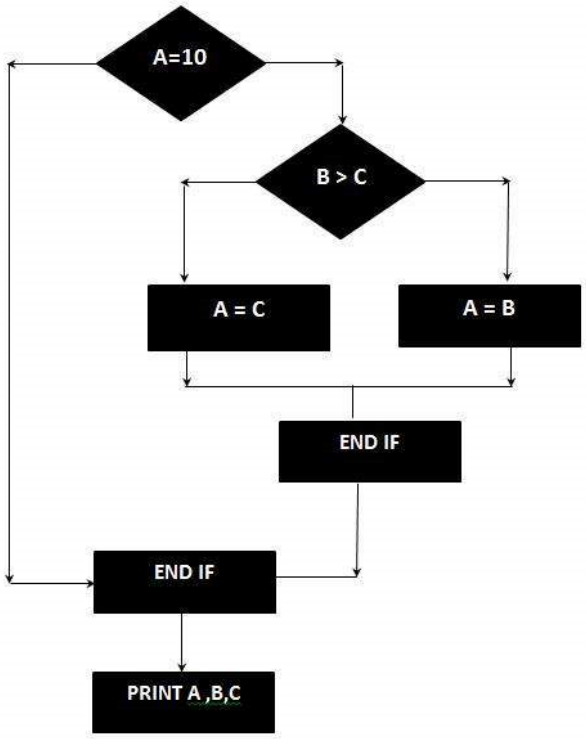
Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand. It can be represented using the below formula:

```
Cyclomatic complexity = E - N + 2*P
where,
  E = number of edges in the flow graph.
  N = number of nodes in the flow graph.
  P = number of nodes that have exit points
```

Example :

```
IF A = 10 THEN
 IF B > C THEN
   A = B
 ELSE
   A = C
 ENDIF
ENDIF
Print A
Print B
Print C
```

FlowGraph:



The Cyclomatic complexity is calculated using the above control flow diagram that shows seven nodes(shapes) and eight edges (lines), hence the cyclomatic complexity is 8 - 7 + 2 = 3

What are metrics,Measures and Indicators?

Metrics represent the different methods we employ to understand change over time across a number of dimensions or criteria. It is often used as a catch-all term to describe the method used to measure something, the resulting values obtained from measuring, as well as a calculated or combined set of measures.
We use the term **measures** when we mean the value measured by whatever mechanism we employ and the term **indicator** for values we combine and use to hint to specific outcomes and trends.

A measure is a number or a quantity that records a directly observable value or performance. All measures are composed of a value (a number) and a unit of measure. The number provides magnitude for the measure (how much), while the unit gives number meaning (what is measured).

- 1,234,567 Pageviews
- 8,901,234 Sessions
- 567,890 Facebook Likes

An indicator is a qualitative or quantitative factor or variable that provides a simple and reliable mean to express achievement, the attainment of a goal, or the results stemming from a specific change. It often aggregates or combines multiple measures in an explicit formula.

- 1M weekly active users
- 1:3 users complete the story
- 23% homepage bounce rate

Explain Black Box Testing in detail.

Black Box testing also called behaviour testing focuses on the functional requirements of the software. That is, black box testing technique enables you to derive sets of input condition that will fully exercise all functional requirement for a program. Black box testing is not an alternative to white Box testing. Rather, it is a complementary approach that is likely to uncover a different class of error than white box methods.

Black box testing attempts to find error in the following categories:

- incorrect or missing function
- interface errors
- errors in data structure or external database access
- behaviour or performance error
- initialization and termination error.

Black box testing tends to be applied during later stage of testing. Because black box testing purposely disregards control structure, attention is focused on the information domain. Tests are designed to answer the following questions

- How is functional validity tested?
- How are system behaviour and performance tested?
- How are the boundaries of a data class isolated?
- what cases of input will make good test cases?
- what data rates and data volume can be the system tolerate?

By applying black box techniques, you derive a set of test cases that satisfy the following criteria:

[redacted]

Date
Page 22

- Test cases that reduce, by a count that is greater than one, the number of additional test cases that must be designed to achieved reasonable testing.
- Test cases that tell you something about the presence or absence of classes of errors rather than an error associated only with the specific test at hand.

What are the different quality factors used to measure software quality.

Correctness:

Correctness is the extent to which a program satisfies its specifications.

Reliability:

Reliability is the property that defines how well the software meets its requirements.

Efficiency:

Efficiency is a factor relating to all issues in the execution of software; it includes considerations such as response time, memory requirement, and throughput.

Usability:

Usability, or the effort required locating and fixing errors in operating programs.

Portability:

Portability is the effort required to transfer the software from one configuration to another.

Reusability:

Reusability is the extent to which parts of the software can be reused in other related applications.

Maintainability:

Maintainability is the effort required to maintain the system in order to check the quality.

Testability:

Testability is the effort required to test to ensure that the system or a module performs its intended function.

Flexibility:

Flexibility is the effort required to modify an operational program.

- **Correctness –**
  The extent to which a software meets its requirements specification.
- **Efficiency –**
  The amount of hardware resources and code the software, needs to perform a function.
- **Integrity –**
  The extent to which the software can control an unauthorized person from the accessing the data or software.
- **Reliability –**
  The extent to which a software performs its intended functions without failure.
- **Usability –**
  The extent of effort required to learn, operate and understand the functions of the software.

- **Maintainability –**
  The effort required to detect and correct an error during maintenance phase.
- **Flexibility –**
  The effort needed to improve an operational software program.
- **Testability –**
  The effort required to verify a software to ensure that it meets the specified requirements.
  - **Portability –**
    The effort required to transfer a program from one platform to another.
  - **Re-usability –**
    The extent to which the program's code can be reused in other applications.
  - **Interoperability –**
    The effort required to integrate two systems with one another.

Write a short note on Integration testing.

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration test approaches – There are four types of integration testing approaches. Those approaches are the following:

1. Big-Bang Integration Testing – It is the simplest integration testing approach, where all the modules are combined and the functionality is verified after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing is very expensive to fix.

Advantages:

- It is convenient for small systems.

Disadvantages:

- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
- High risk critical modules are not isolated and tested on priority since all modules are tested at once.

2. Bottom-Up Integration Testing – In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.

Advantages:

- In bottom-up testing, no stubs are required.
- A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:

- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystems.

3. Top-Down Integration Testing – Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet integrated. In this integration testing, testing takes place from top to bottom. First, high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

Advantages:

- Separately debugged module.
- Few or no drivers needed.
- It is more stable and accurate at the aggregate level.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

4. Mixed Integration Testing – A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches.

Advantages:

- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.
-

Disadvantages:

- For mixed integration testing, it requires very high cost because one part has Top-down approach while another part has bottom-up approach.
- This integration testing cannot be used for smaller systems with huge interdependence between different modules.

Discuss different direct and indirect measures to measure quality of software.

There are **direct** and **indirect** quality measures. The direct quality measures need to be determined as piece of the activities to set quality objects, if such objects are quantified.
In many cases direct quality measures cannot be obtained until it is too late. For instance, for safety critical systems, post-accident measurements provide a direct measure of safety. But because of the colossal harm related to these accidents, we are trying to do all possible to keep off such situations.

In order to control and monitor safety assurance activities different indirect measurements can be used.

There is also an growing price of correcting software bugs late instead of fixing them on early stages in common, because a hidden trouble may lead to other related defects, and the longer it stays not identified, the further removed it is from its root causes, thus making the identification of it even more complicate. Consequently, there is a powerful motive for early indicators of quality that measure quality indirectly.

Indirect quality measures may be used in different quality models to assess and forecast quality, through their established relations to direct quality measures based on historical data or data from other sources.

Consequently, it is necessary to select proper measurements, both direct and indirect, and models to provide quality assessment and feedback.

Discuss different Software Quality Measures.

1. **Code Quality –** Code quality metrics measure the quality of code used for the software project development. Maintaining the software code quality by writing Bug-free and semantically correct code is very important for good software project development. In code quality both Quantitative metrics like the number of lines, complexity, functions, rate of bugs generation, etc, and Qualitative metrics like readability, code clarity, efficiency, maintainability, etc are measured.

2. Reliability – Reliability metrics express the reliability of software in different conditions. The software is able to provide exact service at the right time or not is checked. Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).

3. Performance – Performance metrics are used to measure the performance of the software. Each software has been developed for some specific purposes. Performance metrics measure the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.

4. **Usability –** Usability metrics check whether the program is user-friendly or not. Each software is used by the end-user. So it is important to measure that the end-user is happy or not by using this software.

5. **Correctness –** Correctness is one of the important software quality metrics as this checks whether the system or software is working correctly without any error by satisfying the user. Correctness gives the degree of service each function provides as per developed.

6. **Maintainability –** Each software product requires maintenance and up-gradation. Maintenance is an expensive and time-consuming process. So if the software product provides easy maintainability then we can say software quality is up to mark. Maintainability metrics include time requires to adapt to new features/functionality, Mean Time to Change (MTTC), performance in changing environments, etc.

7. Integrity – Software integrity is important in terms of how much it is easy to integrate with other required software's which increases software functionality and what is the control on integration from unauthorized software's which increases the chances of cyberattacks.

8. Security – Security metrics measure how much secure the software is? In the age of cyber terrorism, security is the most essential part of every software. Security assures that there are no unauthorized changes, no fear of cyber attacks, etc when the software product is in use by the end-user.

What are the different types of System Testing?

Types of System Testing:

- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Write a note on Quality Function Deployment

Quality Function Deployment (QFD) is process or set of tools used to define the customer requirements for product and convert those requirements into engineering specifications and plans such that the customer requirements for that product are satisfied.

- QFD was developed in late 1960s by Japanese Planning Specialist named Yoji Akao.
- QFD aims at translating Voice of Customer into measurable and detailed design targets and then drives them from the assembly level down through sub-assembly level, component level, and production process levels.
- QFD helps to achieve structured planning of product by enabling development team to clearly specify customer needs and expectations of product and then evaluate each part of product systematically.

Key steps in QFD :

1. **Product planning :**
    - Translating what customer wants or needs into set of prioritized design requirements.
    - Prioritized design requirements describe looks/design of product.
    - Involves benchmarking – comparing product's performance with competitor's products.
    - Setting targets for improvements and for achieving competitive edge.
2. **Part Planning :**
    - Translating product requirement specifications into part of characteristics.
    - For example, if requirement is that product should be portable, then characteristics could be light-weight, small size, compact, etc.
3. **Process Planning :**
    - Translating part characteristics into an effective and efficient process.
    - The ability to deliver six sigma quality should be maximized.
4. **Production Planning :**
    - Translating process into manufacturing or service delivery methods.
    - In this step too, ability to deliver six sigma quality should be improved.

Explain White Box Testing in detail. .

* White-Box Testing :→

White Box Testing is a testing technique in which software's internal structure, design and coding are tested to verify input, output flow and improve design, ability and security. In white box testing, code is visible to testers so it is also called as clear box testing, open box testing, transparent box testing, code based testing and glass box testing.

It is one of two part of Box testing approach to software testing. Its counter part, Black box testing, involves testing from an external or end-user perspective. On the other hand, white box testing is also engineering is based on the inner working of an application and involves around internal testing.

The term "white box" was used because of the see-through box concept. The clear box or white box name symbolizes the ability to see through the software outer shell (or "box") into its inner workings.

Like this, white box testing involves the testing of the software code for the following:

- Internal security holes.
- Broken or poorly structured paths in the coding processes.
- The flow of specific input through the code
- Expected output.
- The functionality of conditional loops.
- Testing of each statement, object, and function on an individual basis.

Discuss following testing strategies.
i) Unit Testing
ii) Integration Testing
iii) Regression Testing
iv) Smoke Testing
v) Validation Testing
vi) System Testing

1. **Unit Testing** : checks if software components are fulfilling functionalities or not.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

1. **Integration Testing** : checks the data flow from one module to other modules. This kind of testing is performed by testers.

2. **System Testing** : evaluates both functional and non-functional needs for the testing. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

1. Unit Testing

It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.

Example:

a) In a program we are checking if the loop, method, or

function is working fine

b) Misunderstood or incorrect, arithmetic precedence.

c) Incorrect initialization

2. Integration Testing

The objective is to take unit-tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.

Integration testing is of four types: (i) Top-down (ii) Bottom-up (iii) Sandwich (iv) Big-Bang
Example:

(a) Black Box testing:- It is used for validation.

In this, we ignore internal working mechanisms and focus on **what is the output?**.

(b) White box testing:- It is used for verification.
In this, we focus on internal mechanisms i.e. **how the output is achieved?**

3. Regression Testing

Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole component works properly even after adding components to the complete program.
Example :In school, record suppose we have module staff, students and finance combining these modules and checking if on integration of these modules works fine in regression testing

4. Smoke Testing

This test is done to make sure that the software under testing is ready or stable for further testing
It is called a smoke test as the testing of an initial pass is done to check if it did not catch the fire or smoke in the initial switch on.
Example: If the project has 2 modules so before going to the module make sure that module 1 works properly

7. System Testing

This software is tested such that it works fine for the different operating systems. It is covered under the black box testing technique. In this, we just focus on the required input and output without focusing on internal working.
In this, we have security testing, recovery testing, stress testing, and performance testing
Example: This includes functional as well as nonfunctional testing

VALIDATION TEST

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.