-How does data abstraction and data hiding help in modular design?

Certainly! Data abstraction and data hiding are two important concepts in software design that help

to create better quality software.

Data abstraction means focusing on the important features of an object or system, and ignoring the

less important details. In modular design, data abstraction helps to create simple interfaces between

different parts of the software, so that the details of how each part works can be hidden from the

other parts.

Data hiding means hiding the details of how each part of the software works from the other parts, so

that changes made to one part don't affect the others. By limiting access to the internal state and

behavior of each part of the software, data hiding helps to ensure that the software works as

expected, and doesn't break when changes are made to one part.

By using data abstraction and data hiding, software developers can create software that is easier to

understand, modify, and maintain, which makes it more flexible, reusable, and reliable.

-What are the concepts of modular design?

The concepts of modular design include:

Modularity: breaking down a system or process into smaller, independent parts called modules.

Abstraction: focusing on the essential features of a module and ignoring the unnecessary details.

Encapsulation: hiding the internal workings of a module and exposing only a simple interface to other modules.

Cohesion: ensuring that the elements within a module are related to each other and contribute to a common purpose.

Coupling: minimizing the interdependencies between modules to improve flexibility and ease of maintenance.

By applying these concepts, modular design helps to create software that is easier to understand, modify, and maintain, which in turn leads to more flexible, reusable, and reliable software." can you convert it into simple language

-What is cohesion and coupling? How are they related?

Cohesion is the measure of how well different parts of a module or component work together towards a common goal. High cohesion means they work well together, while low cohesion means they may not work well together.

Coupling refers to how dependent one module or component is on another in the system. Low coupling means they are independent, while high coupling means they are closely connected.

Cohesion and coupling are important in software design because they affect how easy it is to understand and modify the system over time.

-Classify cohesiveness and give 1-2 lines explanation for each.

here are several types of cohesiveness in software engineering, including:

Functional Cohesion: This type of cohesion is achieved when the elements of a module perform a single, well-defined function or task. In other words, all the elements in a module are related and work together to achieve a common goal or purpose.

Sequential Cohesion: Sequential cohesion occurs when the elements of a module are related and work together in a specific order or sequence. This type of cohesion is often seen in modules that perform a series of steps or calculations in a specific order.

Communicational Cohesion: Communicational cohesion is achieved when the elements of a module work together to perform a specific task, but do not share data or state information with each other.

This type of cohesion is often seen in modules that communicate with external systems or APIs.

Procedural Cohesion: Procedural cohesion occurs when the elements of a module are related and work together to perform a specific procedure or process. This type of cohesion is often seen in modules that perform a series of steps or calculations that are closely related.

Temporal Cohesion: Temporal cohesion occurs when the elements of a module are related and work together to perform a specific task, but only during a specific time period or under specific conditions. This type of cohesion is often seen in modules that perform periodic or scheduled tasks.

Overall, cohesion is an important concept in software design because it helps to ensure that modules are well-organized, easy to understand, and maintainable over time. By choosing the appropriate type of cohesion for a given module, developers can ensure that the module is focused, efficient, and effective in performing its intended function.

-Classify coupling with 1 or 2 lines explanation each.

Coupling is classified into the following types:

Content Coupling: When one module directly refers to the content of another module, it is called

content coupling.

Common Coupling: When multiple modules share the same global data, it is called common

coupling.

Control Coupling: When a module passes control information to another module, it is called control

coupling.

Stamp Coupling: When a module passes a composite data structure to another module and only a

part of it is used, it is called stamp coupling.

Data Coupling: When two modules communicate through only data parameters or arguments, it is

called data coupling.

Message Coupling: When one module sends a message to another module and does not share its

data directly, it is called message coupling.

By minimizing coupling and choosing appropriate coupling type, software developers can ensure that modules are more flexible, maintainable and easier to understand.

-What is meant by top-down and bottom-up design? Which type of design handles abstraction and which type of design handles refinement?

Top-down and bottom-up design are two different ways of designing software.

In top-down design, we first define the overall structure and functionality of the system and then break it down into smaller, more manageable components. This approach is good for handling the big picture and abstraction.

In bottom-up design, we first design and test individual components separately, and then integrate them together to form the complete system. This approach is good for handling the details and refinement.

Top-down design is like building a house by first planning the architecture and then breaking it down into smaller tasks like constructing walls, installing doors, etc. Bottom-up design is like building a house by first constructing individual components like walls, roofs, etc. and then integrating them together to form the complete house.

-Learn the different type of structure charts. (Go through them) (control hierarchy)

-Difference between top-down and bottom-up DESIGN.

Here are three key differences between top-down and bottom-up design:

Approach: Top-down design is a deductive approach, meaning it starts with the big picture and works

its way down to the details. In contrast, bottom-up design is an inductive approach that starts with

the details and builds up to the big picture.

Focus: Top-down design focuses on abstraction, where the overall system is decomposed into

subsystems, which are further divided into modules or components. Bottom-up design focuses on

refinement, where smaller components are built first and then combined to form larger components.

Control: Top-down design provides greater control over the overall system architecture and can lead to more consistent and predictable results. Bottom-up design provides greater flexibility and allows for more experimentation and exploration.

Both approaches have their strengths and weaknesses, and the choice of which to use often depends on the specific needs and constraints of the project. In general, top-down design is well-suited for large, complex systems with well-defined requirements, while bottom-up design is better for smaller, more experimental systems where the requirements are less clear.

Both types of testing are important to make sure the software works well and doesn't have any problems.

-What are the subsumption relationships of white-box testing coverage?

Subsumption relationships are like a family tree for concepts or classes in a system. The more general or big ideas are at the top, while the more specific or small ideas are at the bottom. This helps organize and simplify complex systems by grouping related ideas together. For example, "mammals" are a general group that includes "dogs". And "dogs" are a general group that includes "terriers". In software design, this idea can be used to create a hierarchy of classes with related properties and methods.

-What is data flow testing?

Data flow testing is a way of testing software that focuses on how data moves through the program.

The goal is to make sure that data is being used and controlled correctly, and to find any errors or problems that might come up. To do this, testers look at the program's source code and figure out where the data comes in, how it's processed, and where it goes out. Then they create tests to make sure that everything is working the way it should be. This helps find mistakes like missing   
information, code that doesn't do anything, and mistakes in calculations. By finding these problems early, developers can make sure the program works correctly before people start using it.

-What is definition-use path? Why this path is very important during testing?

A definition-use path is a sequence of instructions in a program that defines a variable and then uses it in some way. It's important during testing because it helps identify errors in the way a variable is used in a program. By looking at the path between the point where a variable is declared or initialized and the point where it's used, testers can find potential errors like uninitialized variables or incorrect variable assignments. This helps make sure that the program is working correctly and processing data accurately.

-What errors are revealed in data flow testing?

Data flow testing can reveal a variety of errors in software, including:

Uninitialized variables: Variables that are not initialized with a value can lead to unexpected behavior and errors in the program.

Dead code: Code that is never executed during program execution, which can indicate a programming error or unnecessary code that can be removed.

Logic errors: Errors in the program's logic that can lead to incorrect processing of data.

Incorrect variable assignments: Variables that are assigned incorrect values, which can lead to errors in data processing.

Incorrect use of variables in calculations or other processing: Errors in the way variables are used in calculations or other processing can lead to incorrect results.

By identifying these errors through data flow testing, developers can address them before the program is released, improving its quality and reliability.

-Using control flow graph method of data flow testing, find out whether a program has anomalies or doesn't have an anomaly! Refer to the programs used in the slide "data flow testing 1".

pdf me hai. page no of 9, 10, 11 of data flow testing 1

-Show the subsumption relationships in data flow testing.

Diagram yaad kar le pdf se page no 23 of data flow testing 1.

-What are the problems of testing a loop( If a graph (CFG) contains a loop, it has infinite no.of paths and thus complete path coverage not possible and specific path criteria unsatisfactory)

"Testing a loop in a program can present several challenges. One of the primary challenges is that if a control flow graph (CFG) contains a loop, it can have an infinite number of paths. This makes it difficult or impossible to achieve complete path coverage of the program.

In addition, testing individual paths within a loop can be unsatisfactory because it may not reveal all possible behaviors of the program. The behavior of a loop can depend on its input data, the number of times it is executed, and the specific conditions within the loop.

Another challenge is that loops can introduce additional complexity into a program, making it more difficult to identify and test potential errors. For example, a loop may include nested conditional statements or multiple variables that interact in complex ways.

To address these challenges, testers may use techniques such as boundary value analysis,   
equivalence partitioning, and loop testing to identify and test critical inputs and behaviors within a loop. In addition, code reviews and other forms of static analysis can help identify potential errors in loops before testing even begins." convert it in simple language so that i will able to learn it

-How do you test a loop? (Prime path coverage is used for testing loops)

When testing a loop, there are many ways to approach it. One way is to test the input values at the edges of the loop, such as the minimum and maximum values. Another way is to group the input values into similar groups and test one input from each group. Testers can also design test cases that specifically focus on the loop behavior, such as how it handles different input values, how many times it runs, and how it stops. Prime path coverage is a technique that involves finding the most important paths that cover all possible behaviors in the loop, and then creating test cases to ensure that all of those paths are executed. By using a combination of these techniques, testers can effectively test loops and identify potential errors in the program.

-What is simple path and prime path. Through a small CFG having 4 nodes, mention the simple paths and prime paths.

a simple path is a path that doesn't repeat any nodes, while a prime path is a unique path that cannot be broken down into smaller paths.

A prime path in a control flow graph is a path that satisfies two conditions:

It is a simple path, meaning it does not contain any repeated nodes. It is not a subpath of any other simple path in the graph.

In software testing, a simple path is a path through a control flow graph (CFG) that does not include any repeated nodes or edges. A prime path is a path through a CFG that satisfies the following criteria:

It is a simple path.

It cannot be decomposed into simpler paths by removing any node or edge.

In other words, a prime path is a unique, irreducible path through a CFG.

Let's take an example of a small CFG with 4 nodes:

Start   
 / \   
A B   
 \ /

C

Here, we have three simple paths:

Start -> A -> C   
Start -> B -> C   
Start -> C   
And two prime paths:

Start -> A -> C   
Start -> B -> C   
In this example, the prime paths are also simple paths because there are no repeated nodes or edges. However, in more complex CFGs, prime paths may not always be simple paths.

-What is regression testing and where is it used?

Regression testing is a type of software testing that verifies whether changes made to a software application have introduced any new errors or issues. It involves retesting the modified parts of the application, as well as any related functionality that may be affected by those changes.

Regression testing is used primarily during software maintenance and development cycles. It is particularly useful when new features or functionality are added to a software application, or when bug fixes are implemented. The goal of regression testing is to ensure that changes made to the software have not adversely impacted existing functionality or introduced new issues.

Regression testing can be conducted manually or with the help of automated testing tools. It is an important part of the software development process, as it helps ensure that the software application is functioning correctly and reliably.

-What are the different levels of testing? Relate them with the SDLC phases in a clean diagram.

There are generally four levels of testing in software development. They are:

Unit testing: This level of testing involves testing individual units or components of the software in isolation from the rest of the system. Unit testing is typically done by developers and is focused on verifying that each unit or module works correctly.

Integration testing: This level of testing involves testing how different components or modules of the software work together. Integration testing is focused on ensuring that different parts of the system work together correctly.

System testing: This level of testing involves testing the entire system as a whole. System testing is focused on ensuring that the software meets the requirements and works as intended in the context of the whole system.

Acceptance testing: This level of testing involves testing the software with the end users or customers to ensure that it meets their needs and expectations.

Here is how these levels of testing relate to the phases of the software development life cycle (SDLC):

Unit testing: Unit testing is typically done during the development phase of the SDLC.

Integration testing: Integration testing is typically done during the testing phase of the SDLC.

System testing: System testing is typically done during the testing phase of the SDLC.

Acceptance testing: Acceptance testing is typically done during the deployment and maintenance phases of the SDLC.

The following diagram shows how the different levels of testing relate to the different phases of the SDLC:

-What is the difference between validation and verification?

Validation and verification are two terms that are commonly used in software testing. While they are related, they have different meanings:

Verification: Verification is the process of checking whether the software product meets the specified requirements and specifications. It is a process of evaluating the product before it is released to the customer or end-user. It checks whether the product is built according to the requirements and design specifications. Verification is also known as the “Review Process” where a review is conducted to evaluate the intermediate work products such as design, code, or test cases.

Validation: Validation is the process of checking whether the software product meets the customer's or end-user's requirements and expectations. It is a process of evaluating the product after it has been developed to check whether it meets the intended purpose and functionality. Validation is also known as the “Testing Process” where the actual testing is done on the software product to evaluate its performance and behavior.

In simple terms, verification checks whether the product is built correctly, while validation checks whether the right product is built. Verification is focused on meeting the requirements, design, and specifications, while validation is focused on meeting the needs and expectations of the customer or end-user.

Overall, both verification and validation are important processes in software testing, and both should be done thoroughly to ensure the quality and reliability of the software product.

-Name the functional and non-functional tests.   
Functional tests:

Unit testing   
Integration testing   
System testing   
Acceptance testing   
Regression testing   
Non-functional tests:

Performance testing   
Security testing   
Usability testing   
Compatibility testing   
Load testing   
Stress testing   
Scalability testing   
Reliability testing   
Recovery testing   
-> Load, performance, stress, recovery, usability, securty.

-Name the static and dynamic tests.

Static tests are those that are performed without executing the software code. They include:

Reviews: code reviews, walkthroughs, and inspections.

Static analysis: using tools to analyze the code without executing it, such as syntax checkers, code analyzers, and style checkers.

Dynamic tests are those that are performed by executing the software code. They include:

Unit testing: testing individual units or components of the software.

Integration testing: testing how the different components of the software work together.

System testing: testing the entire system as a whole.

Acceptance testing: testing whether the software meets the customer's requirements.

Regression testing: testing whether changes made to the software have introduced any new errors.

Performance testing: testing the performance of the software under different conditions, such as high loads or large data volumes.

Security testing: testing the software for vulnerabilities and weaknesses in security.

Usability testing: testing how easy the software is to use for its intended audience.

-Study the designing of test cases for equivalence partioning, boundary value analysis where you have to givee the valid and invalid classes.

Sure, here is an overview of designing test cases using equivalence partitioning and boundary value analysis.

Equivalence Partitioning:   
Equivalence partitioning is a technique used in software testing where input data is divided into groups that are expected to exhibit similar behavior. These groups are known as equivalence classes.

To design test cases for equivalence partitioning, you should follow these steps:

Identify input data for the system under test   
Divide the input data into separate equivalence classes based on similar behavior   
Choose one input from each equivalence class as a test case   
Ensure that each test case tests a different aspect of the software   
Test both valid and invalid input values   
Verify that the software behaves as expected for each test case   
For example, if you were testing a login form for a website, you could divide the username input into three equivalence classes: usernames that are too short (less than 6 characters), usernames that are within the valid range (6-20 characters), and usernames that are too long (more than 20 characters). You would then choose one input from each class to create test cases.

Boundary Value Analysis:   
Boundary value analysis is a testing technique that involves testing input values at the boundaries between valid and invalid ranges. This is done to ensure that the software behaves correctly at the edge of acceptable input ranges.

To design test cases for boundary value analysis, you should follow these steps:

Identify input data for the system under test   
Determine the valid and invalid input ranges

Choose values at the boundaries of each range as test cases

Test both valid and invalid input values

Verify that the software behaves as expected for each test case

For example, if you were testing a program that accepts integers between 1 and 10, you would choose the values 1, 2, 10, and 11 as test cases, since these are at the boundaries of the valid input range. You would also choose values outside of the valid range, such as -1 and 12, to test how the software behaves for invalid input values.

-Design a test case for state transition table for logging page of an application which locks after 3 user attempts.

Sure, here's an example test case for the state transition table for the logging page of an application that locks after three user attempts:

Test case ID: TC001

Test case description: Verify the correct state transitions for the logging page when the user attempts to log in with valid and invalid credentials, and when the account gets locked after three failed attempts.

Test steps:

Navigate to the logging page of the application.

Enter valid username and password credentials and click the "Log In" button.

Verify that the user is redirected to the application's main page.

Log out from the application.

Navigate to the logging page of the application again.

Enter an invalid username and/or password and click the "Log In" button.

Verify that the user is shown an error message indicating that the credentials are invalid. Repeat step 6 two more times using different invalid credentials.

Verify that the user is shown an error message after the third invalid attempt, indicating that the account is locked.

Attempt to log in again with valid credentials.

Verify that the user is shown an error message indicating that the account is locked and cannot be accessed at this time.

Wait for the specified lockout period to expire.

Attempt to log in again with valid credentials.

Verify that the user is redirected to the application's main page.

Valid classes:

Valid username and password credentials   
Invalid username and/or password credentials   
Third invalid attempt resulting in account lockout   
Account lockout period   
Invalid classes:

None applicable   
Note: This is just an example test case and may need to be customized based on the specific requirements of the application being tested.

-Design the same test case using decision table testing.

Sure, here is an example of designing the test case using decision table testing for the same scenario:

Decision Table:

Conditions Actions   
User attempts < 3 Log in successful User attempts >= 3 Account locked Incorrect password entered Attempt count +1 Correct password entered Attempt count reset Test Cases:

Test Case 1: User attempts < 3, correct password entered

Condition: User attempts < 3, Correct password entered Expected Result: Log in successful   
Test Case 2: User attempts < 3, incorrect password entered

Condition: User attempts < 3, Incorrect password entered Expected Result: Attempt count +1   
Test Case 3: User attempts = 3, correct password entered

Condition: User attempts >= 3, Correct password entered

Expected Result: Attempt count reset   
Test Case 4: User attempts = 3, incorrect password entered

Condition: User attempts >= 3, Incorrect password entered Expected Result: Account locked   
Test Case 5: User attempts > 3, correct password entered

Condition: User attempts >= 3, Correct password entered Expected Result: Account locked   
Test Case 6: User attempts > 3, incorrect password entered

Condition: User attempts >= 3, Incorrect password entered Expected Result: Account locked

-What is behavioral model? And what type of stimuli does it handle? Define the stimuli.

In software engineering, "behavioral" refers to how a software system works in different situations. This includes how it responds to different user actions, how it handles errors, and how it interacts with other parts of the software. By understanding and testing how the software is supposed to behave, developers can make sure it works the way it's supposed to and meets the needs of users.

In software engineering, stimuli refer to any kind of input or event that triggers a response or behavior from a software system. Stimuli can include user inputs, system events, environmental factors, or any other external or internal triggers that affect the behavior of the software. Stimuli are an important consideration in software testing, as testers need to identify and test the various stimuli that a software system may encounter in order to ensure that it behaves correctly in different scenarios.

Positive stimuli: These are inputs or events that are expected to result in correct or desirable behavior from the system or component.

Negative stimuli: These are inputs or events that are expected to result in incorrect or undesirable behavior from the system or component.

Boundary stimuli: These are inputs or events that are at the edge or limit of the input range or specification, and they are used to test the system's ability to handle extreme or exceptional conditions.

Erroneous stimuli: These are inputs or events that contain errors or invalid data, and they are used to test the system's ability to detect and handle errors or exceptions.

-Draw the STD of a photocopier machine; elevator; ATM system.

ye vaala baad ke liye chhod diya.

- Explain the driver stub model for unit testing a model.

"the driver module is one of the module in driver stub model for unit testing. it provides inputs to software components to check whether the functionalities are working as they should.

"stub module is one of module in the driver stub model to unit testing a software component, which is a piece of code, which simulates that how that particular component behaves in different situation under test and stub module can replace a actual component for test" is it true?

Yes, that is correct. A stub module is a piece of code that simulates the behavior of a software component that is being tested. It replaces the actual component and helps to create a controlled testing environment, where inputs and outputs can be predicted and tested. Stubs are used when the actual component is not yet available or when it is not feasible to use the actual component in testing due to complexity, cost, or other constraints. Stubs are commonly used in conjunction with the driver module in the driver stub model for unit testing.

- Describe how clusters are formed in bottom-up integration testing.

Bottom-up integration testing is an incremental testing approach where individual modules are first tested separately and then combined together to form clusters. Clusters are formed by grouping modules that have already been tested and verified as correct.

In bottom-up integration testing, testing begins with the lowest level modules of the system and proceeds upwards. Each module is tested separately using stubs for any lower-level modules that it depends on. Once a module has been tested and verified, it is integrated with other modules that have also been tested and verified, forming a cluster. The process continues upward, with clusters being integrated with other clusters until the entire system has been integrated and tested.

The formation of clusters in bottom-up integration testing is based on the concept of functional dependency. Modules that depend on other modules are tested and integrated together in the same cluster. This approach helps to identify and resolve issues in the system at an early stage, as the focus is on testing and verifying each module individually and integrating it with other modules only after it has been verified to be correct.

Overall, the formation of clusters in bottom-up integration testing helps to ensure that the system is thoroughly tested and verified, and that any issues are identified and resolved as early as possible.

- What is error, failure, defect, bugs , incident, fault?

In software engineering, the following terms are commonly used to describe different types of problems or issues:

Error: An error is a human action or decision that produces an incorrect or unexpected result.

Failure: A failure occurs when a system or component does not perform its intended function.

Defect: A defect is a flaw or imperfection in a software product that causes it to behave in

unexpected or unintended ways.

Bug: A bug is a coding error or mistake that causes a software program to malfunction.

Incident: An incident is an unplanned event or occurrence that disrupts the normal operation of a

software system.

Fault: A fault is a defect or problem in a hardware or software component that causes it to fail or

malfunction.

-Using a payoff table of your choice with the help of decision tree, how can one decide to invest in stocks or mutual funds or bonds?

pdf se padh lunga baad me.

-Why decision tables are also called cause-effect tables?

Decision tables are also called cause-effect tables because they represent a cause-and-effect relationship between the conditions and actions in a system. In a decision table, the conditions represent the various causes or inputs that can affect the behavior of the system, while the actions represent the various effects or outputs that can be produced by the system. The table shows the

combination of conditions that lead to specific actions or outcomes, which makes it easier to understand the logic and behavior of the system.

Therefore, decision tables are useful tools for specifying complex business rules or requirements in a concise and organized way, and they can be used in various phases of the software development life cycle, such as requirements analysis, design, and testing.

-What does a decision table signify?

A decision table is a tool used in software testing and design to map out different possible conditions and their corresponding actions or outcomes. It is a tabular representation of a logical relationship between conditions, actions, and rules.

Each row in the table represents a specific combination of conditions or inputs, while each column represents a specific rule or action. The cells within the table represent the relationship between a given set of inputs and the corresponding output or action that should be taken.

The decision table is designed to help testers and developers understand the complex relationships

between different conditions and rules, and to identify any potential conflicts or inconsistencies in

the system. By mapping out these relationships in a clear and concise way, decision tables can help

to streamline the testing process and ensure that the software behaves correctly under a wide range

of conditions.

-What are the different entries of a decision table?

-What is the use Or utility of decision table?

-\*Example1, Example2, Example3 in decision table pdf.

- What is sliding window project planning?

- Go through all the structure charts including Email server, Ticket booking system, Library System,

etc.