ASSIGNMENT

QUE-1 ⇒ Explain the fundamental concept of Software testing. discuss different types of Software testing and their objectives.

ANS-1 ⇒ The fundamental Concept of Software testing is to evaluate and verify that a software product or application meets its intended purpose. Software testing is a crucial stage in the software development life cycle. It helps to identify and resolve bugs, improve performance, and ensure that the software is user-focused.

Some Key concepts of software testing ⇒

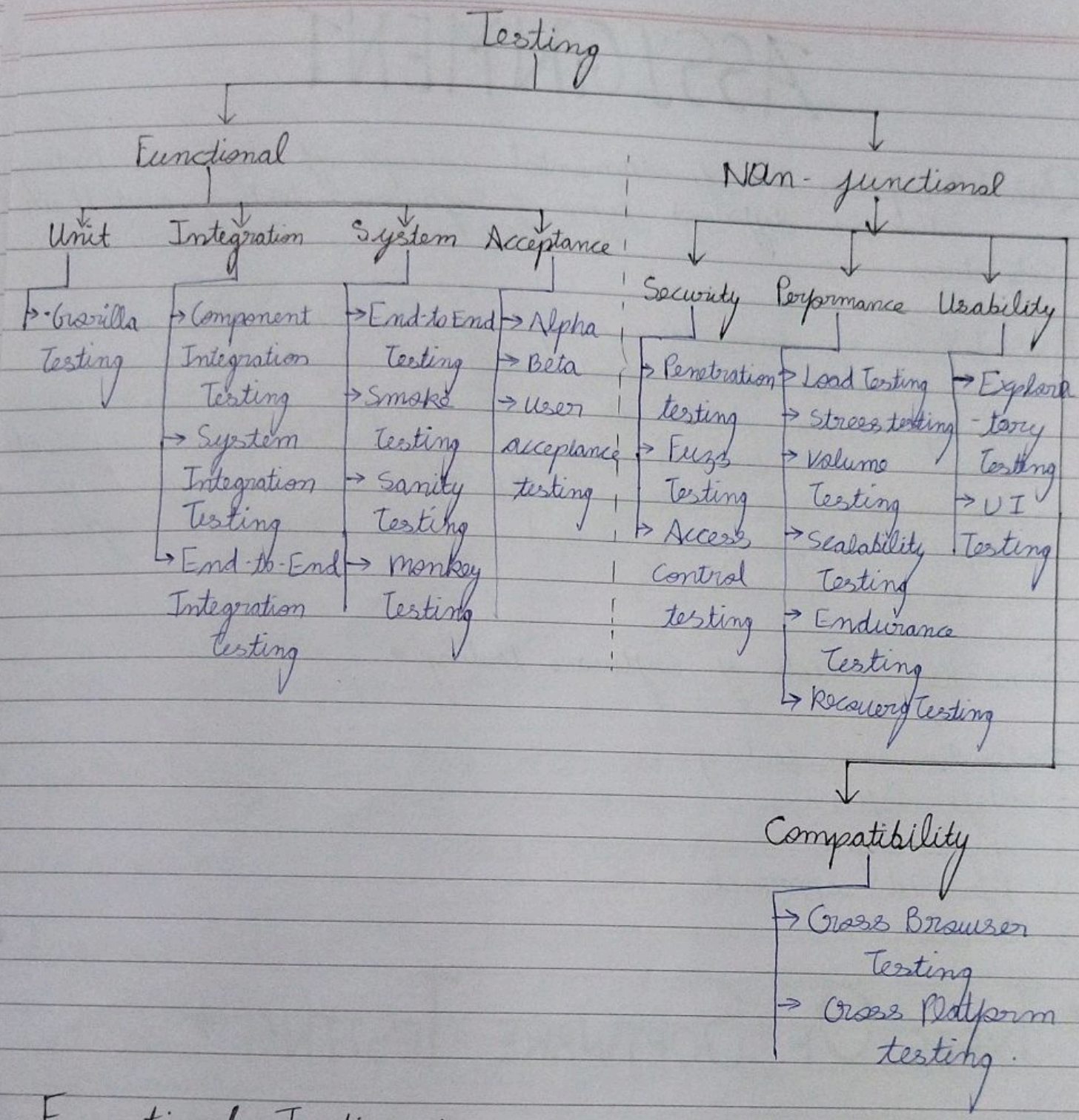
1- Continuous testing
2- Shift-left
3- Shift-right
4- Pyramid approach
5- Oracles.

* TYPES OF SOFTWARE TESTING ⇒

It is divided into two types ⇒

1- Functional Testing

2- Non-Functional Testing



1- Functional Testing ⇒

Focuses on verifying the functionality of the software system. It is a type of testing that is done to ensure that the system works as intended &

Meets the functional requirements specified by the stakeholders.

Functional testing have different types such as ⇒

- Unit Testing ⇒ Performed during the development of an application, this types of testing verifies the correctness of individual units or components of the software.

- Integration Testing ⇒ Performed after unit testing, this type of testing verifies that the interface connection works as expected. It tests for issues like incorrect data transfer or improper functioning when two features are combined.

- System Testing ⇒ This type of testing focuses on the entire software environment and verifies that the system meets specified requirements.

- Acceptance Testing ⇒ Also known as user acceptance testing (UAT), this type of testing ensures that the software meets the goals set in the business requirements. It determines whether the software is acceptable for delivery.

2⇒ Performance Testing ⇒ This types of testing checks how responsive a system is under various circumstances. The goal is to isolate performance bottlenecks so they can be fixed.

1, 2, 3, 5, 4, 7, 6, 8, (10)

2- Non-functional Testing ⇒

Focuses on evaluating the non-functional aspects of the software system. This type of testing includes testing for performance, usability, reliability, scalability, and security.

- Security Testing ⇒ Is a type of software testing that assesses the security of a software application. It helps to identify vulnerabilities and weaknesses in the system and ensure that sensitive data is protected.

- Usability Testing ⇒ Is focused on evaluating the user interface and overall user experience of a software application or system. It involves testing the software with real users to assess its ease of use, learnability, efficiency, and overall user satisfaction.

- Compatibility Testing ⇒ evaluates the compatibility of a software application or system with different hardware, software, operating systems, browsers and other devices or components.

QUE. 2 ⇒ Differentiate the testing and debugging. Explain how these activities complement each other in in the software development phase. Discuss the common debugging techniques used to identify and resolve the defect.

★ANS-2★	TESTING	DEBUGGING
Definition	An activity carried out by testers to find out defect in the software.	An activity carried out by the deployment team after the test report is received.
Purpose	• Identifying & fixing bugs • Ensures reliability and quality performance of the Software.	• Reporting Error Conditions immediatly • Minimising useless and distracting information.
Process	It is the process to identify the failure of code.	It is the process of fixing mistakes in the code.
Programming Knowledge	Most testing requires no Knowledge of the source code.	Debugging requires a proper understanding of the source code.
Performed By	A tester performs testing. Testing can be carried out both by insiders and by outsiders.	Debugging is performed by a programmer. Only an insider does the debugging.
Automation	Testing can be manual or automated.	Debugging is always manual.
Initiation	Testers can only initiate	Debugging begins after

	testing after developers write the code.	a failed test case execution.
Approaches used	• Black Box testing • White Box testing • Unit Testing	• Brute Force • Induction • Deduction

◆ Debugging and testing are complementary processes that ensure software programs run as they should. After writing a complete section or part of a code, programmers test to identify bugs and errors. Once bugs are found, coders can begin the process of debugging and work towards ridding software of any errors.

◆ SOME COMMON DEBUGGING TECHNIQUES ◆

1- Backtracking ⇒ This technique involves starting at the point where the program gives an incorrect result and retracing steps in reverse. It's especially useful when the cause of the bug is unclear

2- Rubber ducking ⇒ This technique involves ~~starting at the~~ explaining the problem or code out loud to an inanimate object, like a rubber duck. This can help you clarify your thoughts and find new perspectives.

3- Print debugging ⇒ This technique involves inserting print statements into the code

to display the values of variables and the state of the program.

4- Cause elimination ⇒ This technique uses deductive techniques to identify possible reasons for the defect.

5- Program slicing ⇒ This technique divides the software into more manageable sections based on variables.

6- Comparing desing vs. Code ⇒ This technique involves comparing coded components with design files to catch discrepancies early.

7- Source code analyzers ⇒ These tools can help with debugging by analyzing security, common code errors, and complexity.

* Other debugging techniques include :-

- Understanding the problem
- Using debugging tools
- Breakpoints and stepping
- Binary Search
- Log analysis
- clustering bugs
- Static analysis
- Remote debugging
- Post-mortem debugging.

Que 3 → Compare and contrast software testing and h/w testing. Discuss the challenges and considerations associated with each type of testing.

Software Vs Hardware Testing

Software testing and hardware testing are two distinct approaches to ensure the quality and reliability of digital products. While both types of testing share some similarities, they differ significantly in scope, environment, approach and focus.

Software Testing :-

- Focuses on verifying and validating s/w functionality, usability and performance.

- Involves testing s/w applications, systems or components on various platforms, including client-server, web and mobile.

Challenges :-

- Complexity of s/w architecture and dependencies.
- Difficulty in reproducing rare or intermittent issues.
- Limited visibility into hardware components behavior.
- Higher cost due to the need for specialized tools and expertise.

Considerations :-

- Test data management and generation.
- Test environment setup and maintenance.
- Resource allocation for testing personnel and infrastructure.

HARDWARE TESTING :-

- Focuses on verifying and validating hardware components' behavior, performance and reliability.
- Involves testing individual components, such as microcontrollers, sensors, or actuators or entire systems like embedded devices or IoT devices.

Challenges :-

- Limited visibility into s/w behavior and interactions.
- Difficulty in simulating complex system interactions.

- Higher risk of physical damage or destruction during testing.
- Limited scalability for testing large numbers of identical devices.

Considerations :-

- Test fixture design and construction.
- Power supply and signal integrity management.
- Environmental control and temperature management.

KEY DIFFERENCE :-

Software testing focuses on the digital aspects of a product, while hardware testing focuses on the physical component.

Que 4 → Explain the difference b/w testing and verification. Discuss the role of verification and validation in ensuring the quality of s/w products.

Ans Verification :- It ensures that the s/w is build acc. to its specifications, design & requirements. It involves :-

- Reviews and Inspections of documents, design and code.
- Code analysis and walkthroughs.
- Checking compliances with specifications, standards and guidelines.

Testing :- Testing, on the other hand, ensures that the s/w works as intended and meets the end-user needs and expectations.

Testing can be further divided into two categories-

- Validation - Ensures that the right product is built, meeting the end-user needs and expectations.
- Verification - Ensure that the product is built correctly, acc. to specifications and design.

⇒ Role of Verification & Validation in Ensuring Quality :-

Verification and validation play crucial roles in ensuring the Quality of s/w products :-

Together, Verification & Validation

1. Prevent Defects :- Verification helps prevent defects by ensuring the development process is correct, while validation detects any defects that may have slipped through.

2. Ensure Quality :- Verification & Validation together ensure that the s/w meets the required quality standards, is reliable and functions as intended.

3. Reduce Rework :- By identifying and addressing defects early, verification and validation reduce the need for rework and minimize the overall cost of s/w development.

In summary, verification ensures that the s/w is built correctly acc. to specifications, while validation ensures that the s/w meets its intended purpose & user expectations. Both processes are essential for delivering high-quality s/w product.

Que 5:- Discuss the auditing process of testing & verification. Discuss the imp. of test matrices. Explain various test matrices that can be used to measure the effectiveness and efficiency of testing activities. Provide an example of how these test matrix can be used to improve the testing process.

Ans.

In summary, verification ensures that the s/w is built correctly acc. to specifications, while validation ensures that the s/w meets its intended purpose & user expectations. Both processes are essential for delivering high-quality s/w product.

Que 5:- Discuss the auditing process of testing & verification. Discuss the imp. of test matrices. Explain various test matrices that can be used to measure the effectiveness and efficiency of testing activities. Provide an example of how these test matrix can be used to improve the testing process.

Ans. **Auditing Test Matrix Process**

The auditing process of testing & verification involves :-

1. Planning :- Identifying the scope objectives and criteria for testing & verification.

2. Execution :- Conducting tests & verifications using various methods, such as inquiry, observation, examination re-performance & computer assisted audit techniques (CAATs)

3. Analysis :- Evaluating the results of testing & verification to determine whether controls are operating effectively & efficiently.

4. Reporting :- Documenting findings and recommendations for improvement.

Importance of Test Matrices

A test Matrix :

1. Ensures comprehensive coverage of testing objectives and criteria.
2. Facilitates prioritization of testing activities based on risk & importance.
3. Provides a clear understanding of the testing scope & boundaries.
4. Enables efficient execution & documentation of testing activities.

Various Test Matrices

1. Control Matrix :- Maps controls to specific objectives & criteria, ensuring that each control is tested against its intended purpose.
2. Test Case Matrix :- Organizes test cases by control, objectives and criteria, providing a detailed plan for testing activities.
3. Risk-Based Test Matrix :- Prioritizes testing activities based on risk assessments, focusing on high-risk areas & controls.
4. Functional Test Matrix :- Groups tests by functional areas, such as data processing, security or compliance to ensure comprehensive coverage.

Example of Using Test Matrices to Improve the Testing Process

The auditor indentifies the following objectives :-

1. Data confidentiality.
2. Data integrity.
3. System Availability.

To plan the testing activities, the auditor creates a control Matrix, which maps controls to each objective:

Objective	control	Description
Data confidentiality	Access control	Limit access to sensitive data.
Data Integrity	Data validation	verify data accuracy and completeness.
System Availability	Redundancy	Ensure duplicate systems for failover.

The auditor then creates a Test Matrix, which outlines specific test cases for each control:

Control	Test Case	Description
Access control	1. Verify access permissions.	Check user access right.
Data validation	2. Validate data accuracy	verify data consistency.
Redundancy	3. verify failover	Simulate system failure.

Q⇒ Explain the concept of model based testing. Describe different types of models of using in testing. Provide example of how model based testing.

MODEL BASED TESTING

MBT is a testing approach where models are used to represent the ~~depict~~ desired behaviour of system under test (SUT).

→ These models serve as an abstract representation of system's functionality & test cases.

→ The models may describe the systems's inputs, outputs and the states it can be in.

Key Features in MBT:

☆ Automation

Test cases are automatically generated based on the model, reducing manual effort.

☆ Efficiency

MBT can generate a high no. of tests, which increases coverage while reducing redundancy.

* Abstract Representation

It allows for an abstract view of the system, focusing on essential aspects rather than implementation details.

Types of MBT

(i) Finite State Machines

→ Represent system using set of states & its transitions b/w those states, triggered by inputs.

→ commonly used in defined states systems

eg ⇒ Vending Machine model

(2) State Charts

→ An extension of finite state machines with more complex state transitions, including hierarchy, parallelism & timing constraints

→ useful for systems with more detailed interaction behaviour.

3. **Decision Tables**

→ Tabular representation where conditions & actions are listed.

→ useful for systems with complex decision logic

eg ⇒ Tax calculation system.

4. **Unified Modeling Language**

→ UML diagrams like use case, sequence & activity diagrams represent interactions & behaviours within a system.

eg ⇒ Diagram showing interaction b/w different modules of web application for processing payments.

5. **Markov Models**

→ Represents systems with probabilistic behaviour.

→ It depict transitions b/w states with certain probabilities.

eg ⇒ mobile Network

Example of MBT

Consider a login system with following requirements ⇒

① User can either log in successfully, fail due to incorrect credentials, or get locked out after multiple failed attempts.

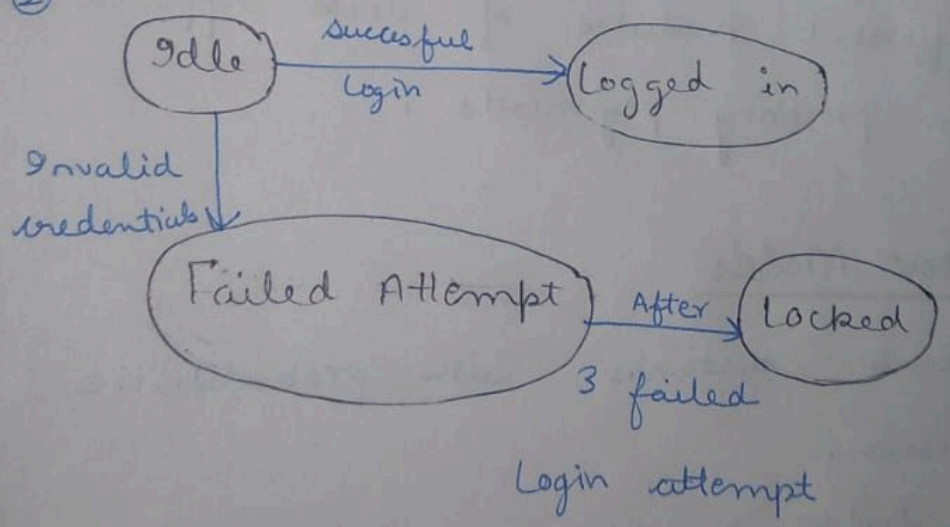
② Login system has states like Idle, Logged in, Failed Attempt & Locked.

(i) Create a model

States ⇒ Idle, Logged in, Failed Attempt, Locked.

Transitions:

①



systems with more detailed interaction behaviour.

ASSIGNMENT

Ques1 - Differed between static and dynamic analysis in software testing. Discuss advantage & disadvantage of each approach. Provide examples of tools and techniques used for static and dynamic analysis.

Ans. These two techniques are complementary and often used together for comprehensive software testing.

Aspect	Static Analysis	Dynamic Analysis
Definition	Examination of code without executing it.	Involves running the program and analyzing its behavior.
Execution	Code is analyzed without being run.	Requires program execution to monitor and tests its behavior.
Purpose	Identifies syntax errors, coding standards, and structural issues.	Detects runtime errors, memory leaks, performance issues etc.
Tools used	Linters, compilers, static code analyzers (eg, SonarQube).	Profilers, debuggers, testing frameworks (eg. JUnit, Selenium).
Type of Testing	Primarily white-box testing.	Can include both white-box and black-box testing.
Coverage	Focuses on code quality, security vulnerabilities, and logic.	Focuses on functional correctness, performance & behavior under various conditions.
Efficiency	Helps identify issues early, before execution	Detects issues that only occur during execution.

Example Issues detected	unused variables, code smells, security vulnerabilities.	memory leaks, crashes, incorrect output, performance bottlenecks.
Complexity	Relatively faster and simpler.	Requires more resources and setup as the program needs to be executed.

* Advantages and disadvantages of <u>static Analysis</u>.

1. Advantages -
   - Early detection of errors.
   - No execution needed
   - Enforces coding standards
   - Security vulnerability detection
   - Broad code coverage.

2. Disadvantages -
   - cannot detect runtime Issues.
   - May Produce False Positives / Negatives
   - Limited Context Awareness
   - Dependent on code Quality.
   - Performance not analyzed.

* Advantages and disadvantages of Dynamic Analysis.

1. Advantages -
   - Finds Runtime errors.
   - Analyzes actual behaviour.
   - Identifies Performance Bottlenecks.
   - can simulate user interactions.
   - useful for functional & integration testing.

2. Disadvantages -
   - Requires Program execution
   - Partial coverage
   - Harder to automate.
   - May miss Early stage Bugs.
   - Resource Intensive

* Static Analysis Tools and Techniques

Tools - SonarQube, Pylint, FindBugs/SpotBugs, checkstyle, EShint, Coverity, Bandit, clang static Analyzer

Techniques - Control Flow Analysis, Data flow Analysis, Pattern-Based Analysis.

* Dynamic Analysis Tools and Techniques

Tools - JUnit, Selenium, Valgrind, Appium, LoadRunner, Dynatrace, AFL (American Fuzzy Lop), JMeter, GDB (GNU Debugger).

Techniques - Unit Testing, Integration Testing, Load/Stress Testing, Fuzz Testing, Profiling.

Ques2. Discuss the concept of control flow graph, state model and data flow based on testing. Explain how these models can be used test cases and analyze the behaviour of a software system.

Ans. 1. Control Flow Graph (CFG) in Testing -

A control Flow Graph (CFG) is a graphical representation of all paths that might be traversed through a program during its execution. Each node in the graph represents a block of code, and the edges between nodes represent control flow paths (e.g. if-else conditions, loops).

How CFG is used Testing :-

Test Case Design - CFG helps in generating test cases that ensure coverage of all possible paths in the program. This is especially useful for branch testing, where test cases are designed to exercise every branch in conditional structures.

- Path coverage - Testers use CFG to achieve statement coverage, branch coverage or path coverage. By analyzing the control flow, testers can create cases to cover specific paths or ensure that all decisions outcomes (like true/false conditions) are tested.

- Detecting loops and Unreachable Code - CFG make it easier to identify loops, which can be tested for issues like infinite loops, and helps detect unreachable code segments that will never execute.

2. State Model in Testing:-

A state Model represents the behaviour of a software system in terms of states and transitions between those states. Each state represents a specific condition or status of the system, and transitions occur in response to events or conditions.

How State Models are used in Testing:-

- State Transition Testing - This type of testing is performed by modeling the different states a system can be in and the events that cause transitions between those states.

- Coverage of Different States - State models allow testers to focus on testing each possible state and transition, ensuring that all the system's behaviors are exercised.

- Testing Invalid Transitions - In addition to testing valid state transitions, testers can create cases to check how the system behaves when an invalid transition is attempted.

3. Data Flow in Testing -

A data flow model focuses on how data moves through a system, tracking the definition, usage, and destruction of data within the program. It highlights how data is transferred b/w diff parts of a system, showing potential problems like incorrect data flow or misuse of data variables.

How Data Flow Models are used in Testing -

- Data Flow Testing - In this technique, test cases are designed to follow paths that data takes through the system. This includes checking that variables are properly initialized, used, and discarded. The aim is to identify anomalies such as unused variables, incorrect assignments, or premature deallocation.

- Variable coverage - Test cases are created to ensure that every variable in the program is tested through its entire lifecycle (from definition to last use), covering scenarios like defining a variable but never using it (dead variables) or using a variable before it is defined (uninitialized variables).

- Detecting Data Anamalies - Data flow models help in identifying issues such as data races, where multiple threads access shared data, or unexpected changes in data values that lead to inconsistent system behavior.

* Using These Models to Design Test cases and Analyze Software Behaviour -

1. Test Case Design:

- Control Flow Graph - Testers can generate test cases to cover every edge & node in the graph, ensuring that every possible path through the system is exercised.

- State Model - Test cases are created for each possible state transition, including both valid & invalid transitions, helping ensure the system responds correctly to a wide range of inputs & events.

- Data Flow - Test cases focus on how data is processed throughout the system, covering all definitions, uses, and kills of variables. This ensures proper handling of data and helps identify any inconsistencies or security vulnerabilities.

2. Analyzing Software Behavior -

- CFG - helps identify potential bugs such as unreachable code or loops that might ~~be~~ not terminate. It gives insight into the logical flow of the program and whether all conditions have been thoroughly tested.
- State Models provide an understanding of the system's dynamic behaviour under various scenarios.
- Data flow models highlight how data is manipulated within the system, making it easier to find bugs related to data usage, like memory leaks, uninitialized variables, or improper variable scope.

Control	Test Case	Description
Access Control	1- Verify access permission	check user access right.
Data Validation	2- Validate data accuracy.	Verify data consistency
Redundancy	3- Verify failover	Simulate system failure.

QUE-6 ⇒ Explain the concept of execution history in software testing. Discuss the various techniques used to analyze execution history and provide examples of how execution history data can be used to improve testing effectiveness and efficiency.

ANS-6 ⇒ Execution history in software testing refers to the detailed records of test execution activities over time. It includes information about which tests were run, their outcomes (Pass or fail), the environment in which they were executed, and any associated defects or issues encountered during testing. This history is crucial for several reasons:

1- Traceability ⇒ It helps track changes in the software and their impact on existing tests, ensuring that all functionalities are verified against modifications.

2- Regression Testing ⇒ A historical record allows testers to identify which tests need to be rerun when changes are made, facilitating effective regression testing.

3- Analysis and Reporting ⇒ Execution history aids in analyzing trends in test results, identifying areas of the software that are problematic, and providing insights for improving test processes.

4- Audit and Compliance ⇒ In regulated industries, maintaining an execution history can be essential for compliance purposes, providing evidence that testing has been conducted thoroughly.

♦ TECHNIQUES USED TO ANALYZED EXECUTION HISTORY

Analyzing execution history in software testing involves several techniques that help derive insights and improve the testing process. Here are some common techniques :-

1- Trend Analysis ⇒ Examining trends over time in pass / fail rates can highlight improvements or regressions in software quality.

2- Defect Analysis ⇒ Analyzing defects based on execution history can help identify

frequently failing tests or high-risk areas of the applicati

3- Root Cause Analysis (RCA) ⇒ When tests fail, RCA techniques (like the "5 Whys" or fishbone diagrams) can help determine underlying causes, enabling teams to address issues at their source.

4- Code Coverage Analysis ⇒ Comparing execution history with code coverage reports helps identify untested areas of the codebase, ensuring comprehensive testing.

5- Test Execution Metrices ⇒ Metrices such as test execution time, the numbers of tests run and failure rates provide quantitative data to assess the effectiveness of the testing process and identify bottlenecks.

- Examples of how execution history data can be used to improve test effectiveness & efficiency :-

Execution history data can significantly enhance test effectiveness and efficiency across various testing methodologies. Here are som examples:-

1- Prioritization of Test cases ⇒ Analyze past execution results to identify frequently failing tests. Focus on these areas firs in subsequent test cycles, ensuring critical fendri

-alities are tested more rigorously.

2- Performance Trends ⇒ Track performance test results over time to identify trends that may indicate issues before they become critical, enabling proactive measures.

3- Automated Test improvement ⇒ Analyze the execution history of automated tests to refine test scripts, address flaky tests, & improve the overall reliability of automation efforts.

4- Regression Testing ⇒ Historical execution data can guide regression testing efforts by highlighting which features have changed or are likely to be affected by new code.

-alities are tested more rigorously.

2- Performance Trends ⇒ Track performance test results over time to identify trends that may indicate issues before they become critical, enabling proactive measures.

3- Automated Test improvement ⇒ Analyze the execution history of automated tests to refine test scripts, address flaky tests, & improve the overall reliability of automation efforts.

4- Regression Testing ⇒ Historical execution data can guide regression testing efforts by highlighting which features have changed or are likely to be affected by new code.

QUE-7 ⇒ Discuss various strategy for test cases. Explain its advantages and disadvantages and provide a example of specific test cases generation techniques.

ANS-7 ⇒ When designing test cases, several strategies can be employed, each with its own advantages and disadvantages, Here are some common strategies:

1- Functional Testing Strategies ⇒

★ **EQUIVALENCE PARTITIONING ⇒** Divide input data into partitions based

on expected output.

Advantages ⇒ Reduces test cases, increases test effectiveness,
Disadvantages ⇒ May miss boundary errors, limited coverage.

★ **BOUNDARY VALUE ANALYSIS** ⇒ Test cases focus on input data boundaries.

Advantages ⇒ Detects boundary errors, ensures robustness.
Disadvantages ⇒ May not cover all input data.

2- Structural Testing Strategies ⇒

★ **STATEMENT COVERAGE** ⇒ Test cases ensure every statement is executed.

Advantages ⇒ Ensures code coverage, detects errors in code.
Disadvantages ⇒ May not cover all logic paths.

★ **DECISION COVERAGE** ⇒ Test cases ensure every decision is executed.

Advantages ⇒ Ensures decision coverage, detects errors in logic.
Disadvantages ⇒ May not cover all code paths.

3- Error-Based Testing Strategies ⇒

★ **ERROR GUESSING** ⇒ Test cases based on experience and intuition.

Advantages ⇒ Detects unexpected errors, improves tester's skills.
Disadvantages ⇒ Limited coverage, not systematic.

★ **FAULT INJECTION** ⇒ Intentionally introduce errors to test error handling.

Advantages ⇒ Detects errors in error handling, improves robustness.
Disadvantages ⇒ Requires significant resources.

4- Exploratory Testing Strategies ⇒

★ **SESSION-BASED TESTING** ⇒ Testers explore software without pre-defined test cases.

Advantages ⇒ Detects unexpected errors, improves tester's skills.
Disadvantages ⇒ Limited coverage, not systematic.

★ **FREESTYLE TESTING** ⇒ Testers explore software without constraints.

Advantages ⇒ Flexible, detects unexpected errors.
Disadvantages ⇒ Limited coverage, not systematic.

5- Model-Based Testing Strategies ⇒

★ **UML-BASED TESTING** ⇒ Test cases generated from UML models.

Advantages ⇒ Ensure coverage of complex systems.

automatic test case generation.
Disadvantages ⇒ Requires modeling expertise.

6- RISK-Based Testing Strategies ⇒

★ **PRIORITY- BASED TESTING** ⇒ Test cases prioritized based on risk and business value.

Advantages ⇒ Ensures high-risk areas are tested, efficient.
Disadvantages ⇒ May miss low-risk errors.

7- Other Strategies ⇒

★ **BLACK BOX TESTING** ⇒ Test cases focus on functionality without internal knowledge.

Advantages ⇒ Easy to create test cases, focuses on functionality.
Disadvantages ⇒ May miss internal errors.

★ **WHITE BOX TESTING** ⇒ Test cases focus on internal code and structure.

Advantages ⇒ Ensures code coverage, detects internal errors.
Disadvantages ⇒ Requires programming knowledge.

♦ Here are some specific test case generation techniques along with examples for each :-

1- Equivalence Partitioning ⇒

Ex ⇒ For a function that accepts age (valid range 18-60). partitions could be:

- Valid: 18, 30, 60
- Invalid (below): 17, 0, -5
- Invalid (above): 61, 100.

2- Boundary Value Analysis ⇒

Ex ⇒ using the same age function, test cases could be -

- Lower boundary: 18 (Valid), 17 (invalid)
- Upper boundary: 60 (Valid), 61 (invalid)

3- State Transition testing ⇒

Ex ⇒ A ~~tu~~ turnstile system:

- States: Locked, Unlocked
- Events: Coin inserted, Push.
- Test cases: 1 - Locked + Coin → Unlocked
  2 - Unlocked + Push → locked

4- Error Guessing ⇒

Ex ⇒ For a form input :-

- Test with empty fields
- Test with special characters
- Test with very large inputs