SEMESTER END EXAMINATION

Subject: Software Testing (BT-CS-PE506)

Max. Marks: 100 Time: 3 Hours

Instructions:

1. This paper contains two sections: **Section A** and **Section B**.
2. **Section A** is compulsory (Short Answer Questions).
3. **Section B** contains Descriptive Questions.
4. Assume missing data suitably if any.

SECTION A – Short Answer Questions

(Attempt all questions. Each question carries 2 marks.) **[10 × 2 = 20 Marks]**

Q1. Differentiate between Verification and Validation.

Answer:

- **Verification:** The process of evaluating work-products (documents, code) of a development phase to ensure they meet the specified requirements of that phase. ("Are we building the product right?")
- **Validation:** The process of evaluating software during or at the end of development to determine whether it satisfies specified business requirements. ("Are we building the right product?")

Q2. Define Fault, Error, and Failure.

Answer:

- **Error (Mistake):** A human action that produces an incorrect result (e.g., a typo in syntax).
- **Fault (Defect/Bug):** The manifestation of an error in the software code.
- **Failure:** The deviation of the software's behavior from its expected delivery or service during execution caused by a fault.

Q3. What is the primary goal of Equivalence Class Partitioning (ECP)?

Answer: ECP is a black-box technique that divides the input domain into classes of data from which test cases can be derived. The goal is to reduce the total number of test cases by assuming that if one value in a class works/fails, all other values in that same class will behave similarly.

Q4. Calculate the Cyclomatic Complexity for a graph having 10 Edges, 7 Nodes, and 1 connected component.

Answer: The formula is $V(G) = E - N + 2P$.

- $E = 10$, $N = 7$, $P = 1$.
- $V(G) = 10 - 7 + 2(1) = 3 + 2 = 5$.

The Cyclomatic Complexity is 5.

Q5. What is a "Test Harness"?

Answer: A Test Harness is a collection of software and test data configured to test a program unit by running it under varying conditions. It typically includes stubs (simulating lower-level modules) and drivers (simulating higher-level calling modules) to execute the code in isolation.

Q6. Distinguish between Alpha and Beta Testing.

Answer:

- **Alpha Testing:** Performed at the developer's site by internal teams or potential users under a controlled environment before releasing to external customers.
- **Beta Testing:** Performed by end-users at the customer's site in a real-world environment without developer control, usually preceding the final release.

Q7. Define Regression Testing and when it is performed.

Answer: Regression testing is the selective re-testing of a system or component to verify that modifications (bug fixes or new features) have not caused unintended effects (regressions) in previously working code. It is performed after every code change.

Q8. What is the significance of the "Defect Bash"?

Answer: Defect Bash is an ad-hoc testing event where different stakeholders (developers, testers, users) simultaneously test the application for a fixed duration to find as many defects as possible. It helps uncover remaining bugs quickly before a major release.

Q9. List any two key components of a Test Plan.

Answer:

(a) **Scope/Objective:** What is being tested and what is out of scope.

(b) **Resource/Schedule:** The hardware/software tools required, the team allocation, and the timeline for testing activities.

Q10. Who is a "Test Specialist"? Mention one key skill required.

Answer: A Test Specialist is a professional whose primary role is to plan, design, execute, and report on testing activities. A key skill required is **analytical thinking** to understand complex logic and identify potential edge cases that developers might miss.

SECTION B – Descriptive Questions

Part 1: 5-Mark Questions (Attempt any 4)

(Answer should be structured and concise. Each question carries 5 marks.) **[4 × 5 = 20 Marks]**

Q11. Explain the Testing Maturity Model (TMM) and list its levels.

Answer: The Testing Maturity Model (TMM) is a framework used to assess the maturity of testing processes in an organization. It helps organizations identify their current testing capabilities and provides a roadmap for improvement.

The 5 Levels of TMM:

(a) **Level 1: Initial:** Testing is chaotic and undefined. There is no formal process; testing is often indistinguishable from debugging.

(b) **Level 2: Phase Definition:** Testing is separated from debugging and is defined as a phase that follows coding. Basic planning exists.

(c) **Level 3: Integration:** Testing is integrated into the entire software lifecycle (not just at the end). Test training programs are established.

(d) **Level 4: Management and Measurement:** Testing is a measured and quantified process. Metrics (defect density, test coverage) are used for quality control.

(e) **Level 5: Optimization/Defect Prevention:** The process focuses on defect prevention rather than detection. Tools and methods are continuously optimized.

Q12. Discuss Boundary Value Analysis (BVA) with a suitable example.

Answer: Concept: BVA is a black-box testing technique based on the observation that defects are more likely to occur at the boundaries of input domains rather than at the center.

Rules: For a valid range min to max, test cases should include:

- min
- $min - 1$ (Invalid)
- $min + 1$
- max
- $max - 1$
- $max + 1$ (Invalid)

Example: An input field accepts integers from 1 to 100.

- Valid Boundaries: 1, 100
- Invalid Boundaries: 0, 101
- Valid Adjacent: 2, 99
- *Test Set:* {0, 1, 2, 99, 100, 101}

Q13. Differentiate between Static Testing and Structural (White Box) Testing.

Answer:

Static Testing	**Structural (White Box) Testing**
Involves examining the documentation and code without executing the program.	Involves running the code and analyzing internal logic, paths, and flows.
Techniques: Reviews, Walkthroughs, Inspections.	Techniques: Statement Coverage, Branch Coverage, Path Testing.
Performed early in the lifecycle (Requirements/Design phase).	Performed during Unit and Integration testing phases.
Finds defects in requirements and logic before coding is finished.	Finds runtime errors, logical loops, and broken paths.

Q14. Compare Top-Down and Bottom-Up Integration Testing.

Answer:

- **Top-Down Integration:**
  - Testing begins with the main module and progressively integrates lower-level modules.
  - **Stubs** are used to simulate missing lower-level modules.
  - **Pros:** Main control logic is tested early; prototype is available early.
  - **Cons:** Writing stubs is time-consuming; low-level utilities are tested late.
- **Bottom-Up Integration:**
  - Testing begins with the lowest level modules (drivers/utilities) and moves upward.
  - **Drivers** are used to call the modules being tested.
  - **Pros:** Critical low-level logic is verified first; easier to create test conditions.
  - **Cons:** The system as a whole doesn't exist until the very end; interface defects found late.

Part 2: 10-Mark Questions (Attempt any 6)

(Detailed answer required with diagrams/examples. Each question carries 10 marks.) **[6 × 10 = 60 Marks]**

Q15. Explain the fundamental "Principles of Software Testing" (Axioms).

Answer: Software testing is guided by a set of general principles (often 7) that provide guidelines for testing effectively.

(a) **Testing shows the presence of defects:** Testing can prove that defects exist, but cannot prove that the software is error-free. Even if no defects are found, it is not a proof of correctness.

(b) **Exhaustive testing is impossible:** Testing all combinations of inputs and preconditions is not feasible (except for trivial cases). We must use risk analysis and priorities to focus testing efforts.

(c) **Early testing:** Testing activities should start as early as possible in the SDLC (e.g., requirement reviews) to fix bugs when they are cheapest to resolve.

(d) **Defect clustering:** A small number of modules usually contain most of the defects discovered during pre-release testing (Pareto Principle: 80% of bugs are in 20% of code).

(e) **Pesticide paradox:** If the same tests are repeated over and over, eventually the same set of test cases will no longer find new bugs. Tests must be reviewed and updated.

(f) **Testing is context-dependent:** Testing is done differently in different contexts (e.g., safety-critical software is tested differently than an e-commerce website).

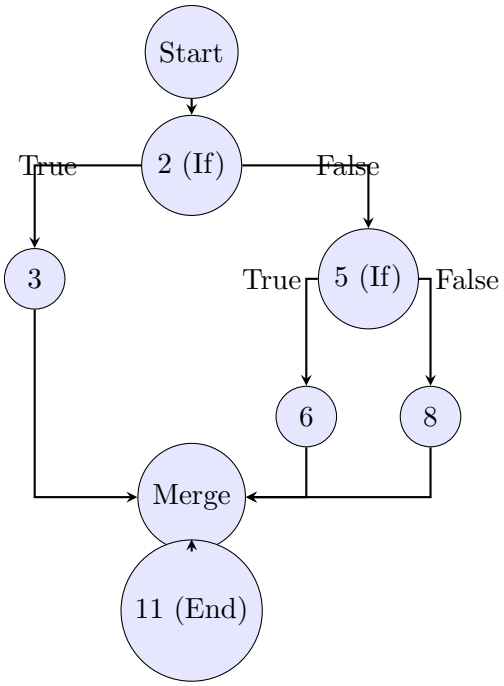
(g) **Absence-of-errors fallacy:** Finding and fixing defects does not help if the system built is unusable or does not fulfill the users' needs and expectations.

Q16. Consider the following C code snippet. Draw the Control Flow Graph (CFG) and calculate the Cyclomatic Complexity using two different methods.

```
1    void calculate_grade(int marks) {
2        if (marks < 0 || marks > 100) {  // Line 2
3            print("Invalid");            // Line 3
4        } else {
5            if (marks >= 50) {           // Line 5
6                print("Pass");           // Line 6
7            } else {
8                print("Fail");           // Line 8
9            }
10       }
11       print("End");                    // Line 11
12   }
```

Answer:

1. Control Flow Graph (CFG):



2. Cyclomatic Complexity Calculation:

Method 1: Formula $V(G) = E - N + 2$

- Nodes (N) = 7 (Start, If1, Action3, If2, Action6, Action8, End/Merge)
- Edges (E) = 9
- $V(G) = 9 - 7 + 2 = 4$

Method 2: Number of Predicate Nodes (P) + 1

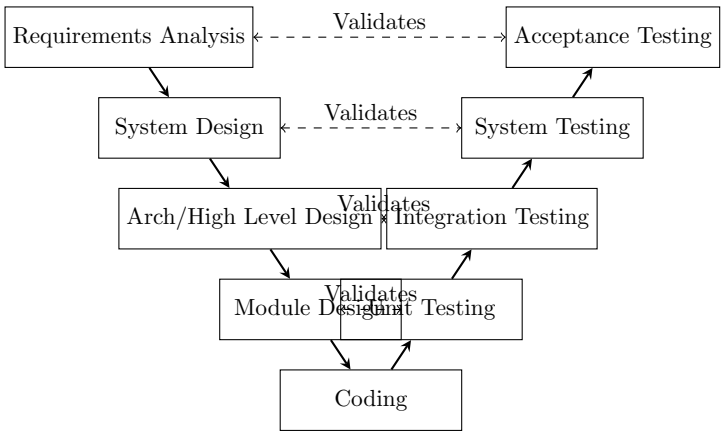
- Predicate nodes are decision points.
- Node 2 is a decision (composite condition '——' often splits into 2 decisions in strict graph theory, but assuming logic level).
- If we treat line 2 ('¡ 0 —— ¿ 100') as two logical checks, V(G) increases. Assuming standard node mapping:

- Decisions: Line 2 (If), Line 5 (If). However, the compound condition '——' effectively adds a hidden path.
- Strict calculation for 'if (A —— B)': This counts as 2 predicates.
- Total Predicates = 3 (First part of OR, Second part of OR, Inner IF).
- $V(G) = P + 1 = 3 + 1 = 4$.

Result: The Cyclomatic Complexity is 4.

Q17. Describe the "V-Model" of software testing. Explain the relationship between development phases and testing levels.

Answer: The V-Model (Verification and Validation Model) illustrates the relationship between each phase of the development life cycle and its associated testing phase. It emphasizes that testing planning should begin alongside development.



Levels Explained:

(a) **Unit Testing:** Verifies individual modules against Module Design. (Done by Developers).

(b) **Integration Testing:** Verifies interfaces between modules against High Level Design.

(c) **System Testing:** Verifies the complete system against System Design/Functional Specs.

(d) **Acceptance Testing:** Verifies the system meets business needs against Requirements. (Done by Clients).

Q18. Explain System Testing. Describe any four types of System Testing (e.g., Performance, Usability).

Answer: System Testing is a level of testing that validates the complete and integrated software product. It is black-box testing performed to evaluate the system's compliance with specified requirements.

Types of System Testing:

(a) **Performance Testing:** Determines how a system performs in terms of responsiveness and stability under a particular workload. It includes Load Testing (normal load) and Stress Testing (extreme load).

(b) **Usability Testing:** Evaluates how user-friendly the application is. It checks navigation, ease of learning, and intuitiveness of the interface.

(c) **Security Testing:** Ensures that the system protects data and maintains functionality as intended. It checks for vulnerabilities like SQL injection, unauthorized access, and encryption failures.

(d) **Recovery Testing:** Forces the software to fail in a variety of ways and verifies that recovery is properly performed. It ensures the system can resume operations after a crash without data loss.

Q19. What is a Test Plan? Detail the structure and components of a standard Test Plan document (IEEE 829).

Answer: A **Test Plan** is a formal document that outlines the scope, objective, strategy, resources, and schedule of a testing effort. It serves as a blueprint for the testing team.

Key Components (IEEE 829 Standard):

(a) **Test Plan Identifier:** A unique ID for the document.

(b) **Introduction:** Summary of the software to be tested and the objectives.

(c) **Test Items:** List of features/modules that are target for testing.

(d) **Features to be Tested:** Specific functionalities in scope.

(e) **Features NOT to be Tested:** Functionalities out of scope (e.g., third-party integrations not ready).

(f) **Approach (Strategy):** How testing will be performed (e.g., automated vs manual, tools used, levels of testing).

(g) **Item Pass/Fail Criteria:** The metrics used to determine if a test item has passed (e.g., "95% of critical test cases passed").

(h) **Suspension/Resumption Criteria:** When to pause testing (e.g., major blocking bug) and when to resume.

(i) **Test Deliverables:** What outputs will be provided (Test cases, Bug reports, Logs).

(j) **Staffing and Training Needs:** Who is on the team and what skills they need.

(k) **Schedule:** Milestones and deadlines.

(l) **Risks and Contingencies:** Potential problems (e.g., server downtime) and backup plans.

Q20. What is Object-Oriented (OO) Testing? How does it differ from procedural testing? Explain Class Testing.

Answer: OO Testing refers to the testing of software built using Object-Oriented paradigms (Encapsulation, Inheritance, Polymorphism).

Difference from Procedural Testing:

- In procedural testing, the unit is a function or subroutine. In OO testing, the unit is typically a **Class**.
- OO testing must address inheritance (testing a subclass requires re-testing inherited methods in the new context) and polymorphism (dynamic binding makes static analysis difficult).

Class Testing:

- Since methods inside a class interact with the class state (variables), they cannot be tested in isolation like simple functions.
- **Approach:**

(a) **Random Testing:** identifying valid sequences of method calls.
(b) **Partitioning:** Grouping states or methods based on functionality.
(c) **State-Based Testing:** Using State Transition Diagrams to ensure the class moves from one valid state to another correctly upon method execution.