STQA Unit-2 Notes By OMG

1. Explain use case testing with one example.

Use case testing is a black-box testing technique that helps to identify test cases from the user's perspective. It focuses on how a user interacts with a system to achieve a specific goal. Use cases describe the step-by-step interactions between the user (actor) and the system.

Example: Online Shopping Cart

Let's consider a simple use case for an e-commerce website: "Add to Cart".

- **Use Case:** Add a product to the shopping cart.
- **Actor:** Customer
- **Goal:** The customer wants to add a selected product to their shopping cart.

Main Success Scenario (Happy Path):

1. The customer browses the product catalog.
2. The customer selects a product.
3. The customer clicks the "Add to Cart" button.
4. The system adds the product to the shopping cart.
5. The system displays a confirmation message, "Product added to cart successfully."
6. The cart icon updates to show the number of items.

Alternative Scenarios (Exceptions/Edge Cases):

- **Scenario 1: Product is out of stock.**
  1. The customer selects a product that is out of stock.
  2. The customer clicks "Add to Cart".
  3. The system displays an error message, "Sorry, this product is currently out of stock."
  4. The product is not added to the cart.
- **Scenario 2: Adding the same product twice.**
  1. The customer adds a product to the cart.
  2. The customer navigates back to the product page and clicks "Add to Cart" again for the same product.
  3. The system updates the quantity of that product in the cart instead of adding it as a new item.
- **Scenario 3: Invalid quantity.**
  1. The customer tries to add a product with a quantity of zero or a negative number.
  2. The system displays an error message, "Please enter a valid quantity."

Use case testing ensures that the system behaves as expected in both normal and exceptional situations, covering the end-to-end user flow.

---

2. Differentiate between Test plan and Test strategy.

Basis for Comparison	Test Plan	Test Strategy
Definition	A detailed document that describes the scope, objective, approach, and resources for a testing project.	A high-level document that defines the organization's approach to software testing.
Scope	Project-specific. A new test plan is created for each project.	Organization-wide or product-line specific. It's a long-term document.
Focus	"What" and "How" of testing for a specific project.	"Why" and "How" of testing at a higher level.
Components	Includes details like test scope, test schedule, resources, test deliverables, and risk management.	Includes testing approach, test levels, test types, tools, and metrics.
Author	Typically written by the Test Lead or Test Manager of the project.	Usually created by the QA Manager or a senior QA professional.
Level of Detail	Very detailed and specific to the project's requirements.	High-level and provides a general framework for testing.
Changes	Can be changed or updated as the project	It is a static document and

	progresses.	is not changed frequently.
Example	A test plan for a new e-commerce website's login feature.	A company's overall strategy for testing all its web applications.

---

3. What are the entry & exit criteria of testing.

Entry Criteria are the conditions that must be met *before* a testing phase can begin. They act as a checklist to ensure that the testing team has everything they need to start testing effectively.

Common Entry Criteria:

- **Test Environment is Ready:** The hardware and software environment for testing is set up and configured.
- **Testable Code is Deployed:** The build has been successfully deployed to the test environment.
- **Test Cases are Ready:** Test cases have been written, reviewed, and are ready for execution.
- **Test Data is Available:** The necessary test data has been created and is available.
- **Required Tools are Available:** All necessary testing tools (e.g., test management, automation tools) are available and working.

Exit Criteria are the conditions that must be met *before* a testing phase can be considered complete. They define the "definition of done" for a testing cycle.

Common Exit Criteria:

- **All Test Cases Executed:** All planned test cases have been executed.
- **Defect Density Reached:** The number of defects found is within an acceptable range.
- **No Critical or Blocker Defects are Open:** All high-priority defects have been fixed and verified.
- **Test Coverage Achieved:** The desired level of code coverage or requirement coverage has been met.
- **Successful Completion of a Full Regression Cycle:** A full regression test has been successfully completed without any major issues.

---

4. Differentiate between verification and validation.

Basis for Comparison	Verification	Validation
Question it answers	"Are we building the product right?"	"Are we building the right product?"
Focus	Checks if the software conforms to its design and specifications.	Checks if the software meets the customer's needs and requirements.
Timing	Done *before* validation, often during the development process.	Done *after* verification, typically on the final product.
Activities	Includes reviews, walkthroughs, inspections, and static analysis.	Includes testing the actual product through various testing techniques.
Involvement	Involves developers, QA team, and peer reviewers.	Involves the testing team and end-users.
Artifacts	Checks documents like requirements, design, and code.	Checks the final, executable software product.
Nature	Static process - does not involve executing the code.	Dynamic process - involves executing the code.
Goal	To prevent defects from being introduced in the first place.	To find defects that were missed during verification.

---

5. Justify: i) Green money 1 cost of prevention. ii) Red money 1 cost of failure.

This question seems to have a typo and likely refers to the "Cost of Quality" concept. Let's interpret it as:

- **Green Money:** The money spent to *prevent* defects (Cost of Prevention).
- **Red Money:** The money spent because of *failures* (Cost of Failure).

i) Justification for "Green Money" (Cost of Prevention):

"Green Money" represents a proactive investment in quality. It includes costs associated with activities aimed at preventing defects from occurring in the first place. Examples include:

- **Quality Planning:** Defining quality standards and processes.
- **Training:** Educating the team on coding standards, testing techniques, and quality processes.
- **Reviews and Inspections:** Conducting code reviews and design reviews to catch issues early.
- **Process Improvement:** Continuously improving development and testing processes.

The justification is that **investing in prevention is cheaper in the long run.** Finding and fixing a defect early in the development lifecycle (e.g., during the design phase) is significantly less expensive than fixing it after the product has been released to customers.

ii) Justification for "Red Money" (Cost of Failure):

"Red Money" represents the reactive cost incurred due to defects and failures. This can be categorized into:

- **Internal Failure Costs:** Costs of defects found *before* the product is delivered to the customer. This includes re-testing, bug fixing, and rework.
- **External Failure Costs:** Costs of defects found *after* the product is delivered to the customer. This is the most expensive type of failure and includes customer support, warranty claims, product recalls, brand damage, and potential loss of customers.

The justification here is that **the cost of failure can be enormous and can even lead to business failure.** Therefore, it is crucial to minimize "Red Money" by investing in "Green Money" (prevention) and "Blue Money" (appraisal/testing).

---

6. Discuss Integration testing and Acceptance testing.

Integration Testing:

- **Purpose:** To test the interfaces between different modules or components of a software application to ensure they work together correctly. It is performed after unit testing and before system testing.

- **Goal:** To find defects in the interaction between integrated units.
- **Approaches:**
    - **Big Bang:** All modules are integrated at once and then tested. This is suitable for small systems.
    - **Top-Down:** Testing starts from the top-level modules and moves downwards. Stubs (dummy modules) are used to simulate lower-level modules.
    - **Bottom-Up:** Testing starts from the lower-level modules and moves upwards. Drivers are used to simulate higher-level modules.
    - **Sandwich (Hybrid):** A combination of top-down and bottom-up approaches.

Acceptance Testing:

- **Purpose:** To determine if the software meets the business requirements and is ready for delivery to the end-users. It is the final phase of testing before the software is released.
- **Goal:** To gain the user's confidence and acceptance of the system.
- **Types:**
    - **User Acceptance Testing (UAT):** Performed by the end-users to validate the software against their business needs.
    - **Business Acceptance Testing (BAT):** Ensures the software meets the business goals and objectives.
    - **Alpha Testing:** Performed by the internal team (developers, testers) at the developer's site.
    - **Beta Testing:** Performed by a limited number of real users in their own environment.

---

7. Analyse test policy & test strategy which is included in test documentation.

Test Policy:

- **What it is:** A high-level document that outlines the organization's overall philosophy and commitment to quality and testing. It defines the "why" of testing for the entire organization.
- **In Test Documentation:** The test policy serves as the guiding principle for all testing activities. It sets the tone and direction for quality. It is a very high-level document and is not part of the project-specific test documentation. It is more of a corporate-level document.

Test Strategy:

- **What it is:** A document that defines the organization's standard approach to testing. It outlines the testing methodologies, tools, and processes to be used across different projects. It addresses "how" the testing will be done in general.

- **In Test Documentation:** The test strategy provides the framework for the project's **Test Plan**. The test plan will derive its approach from the test strategy but will add project-specific details. For example, the test strategy might state that "all web applications will be tested for performance using JMeter," and the test plan for a specific project will then detail the specific performance test scenarios for that application.

Analysis:

- **Hierarchy:** Test Policy -> Test Strategy -> Test Plan.
- **Relationship:** The test policy is the highest-level document, providing the vision for quality. The test strategy translates this vision into a general approach to testing. The test plan then takes the strategy and applies it to a specific project with detailed planning.
- **Importance:** Both are crucial for ensuring consistency and standardization in testing across an organization. They help in setting clear expectations and provide a roadmap for achieving quality goals.

---

8. Define the term: 1) error 2) bug 3) fault and 4) failure 5) Benchmark

1. **Error:** A mistake made by a human (e.g., a developer, business analyst, or system architect). For example, a developer misunderstanding a requirement and writing incorrect code.
2. **Bug (or Defect):** The result of an error. It is a flaw or imperfection in the software code that causes the system to behave in an unintended or incorrect way.
3. **Fault:** An incorrect step, process, or data definition in a computer program. A fault is the representation of an error in the software. It is a static condition in the code.
4. **Failure:** The inability of a software system to perform its required function. A failure occurs when a fault is executed. It is the observable, incorrect behavior of the system.
5. **Benchmark:** A standard or a point of reference against which things may be compared or assessed. In software testing, benchmarking is the process of comparing the performance of a system against a set of standards or a baseline.

---

9. What skills are expected in a good tester?

A good tester should possess a combination of technical and soft skills:

- **Attention to Detail:** The ability to notice even the smallest discrepancies.
- **Analytical and Problem-Solving Skills:** The ability to analyze complex situations, identify the root cause of a problem, and think of creative solutions.
- **Curiosity and a "Testing Mindset":** A natural curiosity to explore the system, question

assumptions, and think about "what if" scenarios.
- **Communication Skills:** The ability to clearly and concisely report defects, communicate with developers and other stakeholders, and write good test documentation.
- **Technical Skills:** Understanding of the software development lifecycle, testing methodologies, and proficiency in using testing tools (e.g., test management, automation, performance testing tools).
- **Domain Knowledge:** Understanding the business domain for which the software is being developed.
- **Time Management and Organization:** The ability to manage multiple tasks, prioritize work, and meet deadlines.
- **Adaptability:** The ability to adapt to changing requirements and technologies.

---

10. Explain the test efficiency and defect rejection.

Test Efficiency:

- **Definition:** A metric used to measure the effectiveness of the testing process. It is the ratio of the number of defects found by the testing team to the total number of defects in the software (including those found by the customer after release).
- **Formula:**
  `Test Efficiency = (Number of defects found in a test phase / Total number of defects in that phase) * 100`

- **Importance:** A high test efficiency indicates that the testing process is effective in finding defects before the software is released to the customer. It helps in improving the quality of the software and reducing the cost of failure.

Defect Rejection:

- **Definition:** The process where the development team rejects a defect reported by the testing team.
- **Reasons for Rejection:**
  - **Not a Defect:** The reported issue is actually the expected behavior of the system.
  - **Duplicate Defect:** The defect has already been reported.
  - **Cannot Reproduce:** The developer is unable to reproduce the defect based on the information provided.
  - **Works as Designed:** The system is working according to its design, even if the tester thinks it's incorrect.
  - **Environment Issue:** The defect is caused by an issue in the test environment, not the code.
- **Importance:** A high defect rejection rate can indicate problems in the testing process,

such as a lack of understanding of the requirements, poor defect reporting, or an unstable test environment. It's important to analyze the reasons for rejected defects to improve the overall quality process.