STQA Unit-1 Notes By OMG

1. SDLC for a Real-Life Application: E-commerce Website

The **Software Development Life Cycle (SDLC)** is a framework that defines the steps involved in the development of a software product. Here's a breakdown of the SDLC for creating an e-commerce website.

Phase 1: Requirement Analysis

This is the initial and most crucial phase. It involves gathering all the necessary information from the client to understand their vision for the product. For an e-commerce site, the requirements would include:

- **User registration and login:** A secure system for users to create accounts and log in.
- **Product catalog:** A comprehensive display of all products with detailed descriptions, images, and prices.
- **Search and filter functionality:** An efficient way for users to find the products they are looking for.
- **Shopping cart:** A feature to allow users to add multiple items before proceeding to checkout.
- **Payment gateway integration:** A secure system to process online payments.
- **Order management system:** A backend system for the admin to manage orders, shipping, and returns.
- **Customer reviews and ratings:** A feature to allow users to provide feedback on products.

Phase 2: Design

Once the requirements are clear, the design phase begins. This involves creating the architectural design and the user interface (UI) and user experience (UX) design.

- **High-Level Design (HLD):** This outlines the overall architecture of the application, including the database structure and the interaction between different modules.
- **Low-Level Design (LLD):** This provides a more detailed design of individual modules, specifying the logic and data structures.
- **UI/UX Design:** This focuses on the look and feel of the website, creating wireframes and prototypes to visualize the user journey.

Phase 3: Development

This is the phase where the actual coding of the application takes place. Developers use the design documents to write the code for the front-end (what the user sees) and the back-end (the server-side logic).

- **Front-end Development:** Creating the user interface using technologies like HTML, CSS, and JavaScript.
- **Back-end Development:** Building the server-side application, database, and APIs using languages like Python, Java, or PHP.

Phase 4: Testing

After the development phase, the application is thoroughly tested to identify and fix any bugs or defects[1].

- **Unit Testing:** Testing individual components of the application.
- **Integration Testing:** Testing the interaction between different modules.
- **System Testing:** Testing the entire application as a whole.
- **User Acceptance Testing (UAT):** The client or end-users test the application to ensure it meets their requirements.

Phase 5: Deployment

Once the application has been tested and approved, it is deployed to a live server, making it

accessible to the public.

Phase 6: Maintenance

After deployment, ongoing maintenance is required to ensure the application continues to function smoothly. This includes fixing any new bugs, updating the application with new features, and providing customer support.

---

2. Pillars of a Quality Management System

A **Quality Management System (QMS)** is a formalized system that documents processes, procedures, and responsibilities for achieving quality policies and objectives. The key pillars of a robust QMS include:

- **Customer Focus:** The primary focus of a QMS is to meet and exceed customer expectations. Understanding the current and future needs of customers is essential for success.
- **Leadership:** Strong leadership is crucial for setting a clear vision and direction for the organization's quality objectives. Leaders must create an environment that encourages employees to achieve these objectives.
- **Engagement of People:** Competent and empowered employees at all levels are the essence of an organization. Their full involvement enables their abilities to be used for the organization's benefit.
- **Process Approach:** A desired result is achieved more efficiently when activities and related resources are managed as a process. This involves understanding and managing interrelated processes as a system.
- **Improvement:** Continual improvement is an ongoing effort to enhance an organization's products, services, or processes. This is a permanent objective of any successful organization.
- **Evidence-based Decision Making:** Decisions based on the analysis and evaluation of data and information are more likely to produce desired results.
- **Relationship Management:** An organization and its external providers (suppliers, contractors, etc.) are interdependent, and a mutually beneficial relationship enhances the ability of both to create value.

---

3. Software Testing and Its Necessity

What is Software Testing?

Software testing is the process of evaluating a software application to identify any gaps, errors, or missing requirements contrary to the actual requirements[2]. The goal is to ensure that the software is free of defects and meets the end-user's needs. This involves executing the software with the intent of finding bugs.

Why is Testing Needed?

Testing is a critical part of the software development lifecycle for several reasons[3]:

- **To Identify and Fix Bugs:** The primary purpose of testing is to find and report defects so they can be fixed before the software is released to the public.
- **To Ensure Quality:** Testing helps to ensure that the software meets the specified quality standards and requirements. A thoroughly tested product inspires confidence in the end-user.
- **To Improve User Experience:** A bug-free and smooth-functioning application leads to a better user experience, which is crucial for customer satisfaction and retention.
- **To Reduce Costs:** Finding and fixing bugs early in the development process is significantly cheaper than fixing them after the software has been released.
- **To Ensure Security:** Testing helps to identify and fix security vulnerabilities that could be exploited by malicious users.

---

4. "Quality" from Different Stakeholder Perspectives

The definition of

"Quality" can vary significantly depending on the stakeholder's perspective in the software development process[4].

- **Customer/End-User:** For the end-user, quality means the software is easy to use, reliable, and performs the intended functions correctly and efficiently. It should be free of errors and meet their specific needs.
- **Developer:** From a developer's point of view, quality often relates to the elegance and maintainability of the code. A quality piece of software is well-documented, easy to understand, and can be easily modified or extended in the future.
- **Project Manager:** A project manager's perspective on quality is often tied to the project's constraints. For them, a quality product is one that is delivered on time, within budget, and meets all the specified requirements.
- **Business Owner/Company:** For the business, quality software is a product that is profitable, enhances the company's reputation, and gives them a competitive advantage in the market. It should be a product that customers are willing to pay for and recommend to others.

---

5. The PDCA Cycle

The

PDCA (Plan-Do-Check-Act) cycle is an iterative four-step management method used in business for the control and continual improvement of processes and products[5].

Plan

In this phase, you identify and analyze the problem or opportunity. You develop hypotheses about what the issues are and decide which ones to test. This involves:

- Defining the problem to be addressed.
- Gathering data.
- Determining the root cause of the problem.
- Developing a plan with clear objectives and success metrics.

Do

This phase involves implementing the plan on a small scale. This allows you to test the proposed solution without causing major disruption. Key activities include:

- Implementing the changes identified in the "Plan" phase.
- Documenting any changes made and collecting data for analysis in the next phase.

Check

In the "Check" phase, you analyze the results of the small-scale test. This involves:

- Monitoring the implemented changes.
- Comparing the results against the expected outcomes.
- Identifying any deviations from the plan.

Act

Based on the results from the "Check" phase, you decide on the next steps.

- If the change was successful, you implement it on a larger scale.
- If the change was not successful, you go back to the "Plan" phase to revise your approach.
- The cycle then repeats for continuous improvement.

---

6. The Relationship Between Quality and Productivity

There is a direct and positive relationship between

Quality and **Productivity**[6]. Historically, it was believed that increasing quality would lead to a decrease in productivity due to the extra time and resources required for quality control.

However, the modern understanding is that higher quality leads to increased productivity in the long run.

Here's how they are related:

- **Reduced Rework:** High-quality processes lead to fewer defects in the final product. This means less time and resources are spent on fixing errors and redoing work, which directly improves productivity.
- **Increased Efficiency:** Focusing on quality often involves streamlining processes and eliminating waste. This makes the entire development process more efficient, allowing more to be produced with the same or fewer resources.
- **Improved Customer Satisfaction:** High-quality products lead to happier customers, which can result in repeat business and positive word-of-mouth, ultimately boosting sales and the company's bottom line.
- **Lower Costs:** While there might be an initial investment in quality assurance, the long-term savings from reduced rework, fewer warranty claims, and increased customer loyalty often outweigh the initial costs.

---

7. Comparison Between Software Testing Tools and Techniques

Software testing tools and **techniques** are both essential for effective software testing, but they serve different purposes[7].

Feature	Software Testing Tools	Software Testing Techniques
Definition	Applications and software used to automate the testing process.	Methodologies and strategies used to design and execute tests.
Purpose	To automate repetitive tasks, improve efficiency, and increase test coverage.	To ensure that the software is tested systematically and thoroughly.

Examples	Selenium, JUnit, TestRail, Jira	Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, State Transition Testing
Nature	Concrete and practical. They are the "what" you use to test.	Abstract and conceptual. They are the "how" you approach testing.
Role in Testing	They execute the tests designed using various techniques.	They provide a framework for creating effective test cases.

In essence, **testing techniques** guide the strategy of what and how to test, while **testing tools** are the instruments that help in executing that strategy more efficiently.

---

8. The Difference Between Testing & Debugging

Testing and **debugging** are two distinct but related activities in the software development process[8].

Basis for Comparison	Testing	Debugging
Goal	To find defects in the software.	To find the root cause of a known defect and fix it.
Process	Planned and systematic execution of the software with the intent of finding errors.	An unplanned and often ad-hoc process that starts after a bug has been identified.
Performed by	Primarily done by testers.	Primarily done by developers.

Timing	Occurs throughout the software development lifecycle, but mainly after the development phase.	Occurs after a bug has been reported by the testing team.
Outcome	A list of identified bugs and a report on the overall quality of the software.	A bug-free piece of code.

In short, **testing finds the bugs, and debugging fixes them.**

---

9. "Total Quality Management" Principles of Continual Improvement

Total Quality Management (TQM) is a management approach to long-term success through customer satisfaction. A core principle of TQM is

continual improvement[9], which is based on the following ideas:

- **Customer-Centered:** The customer is the ultimate judge of quality. All improvements should be aimed at increasing customer satisfaction.
- **Total Employee Involvement:** All employees participate in working toward common goals. This fosters a sense of ownership and responsibility for quality.
- **Process-Centered:** A fundamental part of TQM is a focus on process thinking. A process is a series of steps that take inputs from suppliers and transforms them into outputs that are delivered to customers.
- **Integrated System:** All departments and functions within an organization should be interconnected and work together to achieve quality goals.
- **Strategic and Systematic Approach:** A strategic plan that includes quality as a core component is essential for achieving an organization's vision.
- **Fact-Based Decision Making:** Decisions for improvement should be based on data and analysis rather than on opinions or assumptions.
- **Communication:** Effective communication is vital for maintaining morale and motivating employees at all levels.

---

10. Software Quality Control for Space Research

Software Quality Control (SQC) in space research is of paramount importance due to the mission-critical nature of the software[10]. A failure in software can lead to catastrophic consequences, including mission failure and loss of expensive equipment. A plan for SQC in this domain would include:

- **Rigorous Requirements Analysis:** Every requirement must be meticulously documented, analyzed, and traced throughout the development lifecycle. There is no room for ambiguity.
- **Formal Design and Modeling:** The software architecture and design must be formally specified and modeled. This allows for early detection of design flaws through simulations and formal verification techniques.
- **Stringent Coding Standards:** Adherence to strict coding standards is mandatory. This includes rules for code style, commenting, and the use of specific language features to avoid common programming errors.
- **Exhaustive Testing and Validation:**
    - **Static Analysis:** Automated tools are used to analyze the source code without executing it to find potential defects.
    - **Dynamic Analysis:** The software is executed with a wide range of inputs and under various conditions to identify any runtime errors.
    - **Hardware-in-the-loop (HIL) Simulation:** The software is tested with actual hardware components in a simulated environment to ensure seamless integration and performance.
    - **Fault Injection Testing:** Deliberately introducing faults to test the system's error-handling capabilities.
- **Independent Verification and Validation (IV&V):** An independent team, separate from the development team, is responsible for verifying that the software meets all requirements and validating that it is fit for its intended purpose.
- **Traceability Matrix:** A traceability matrix is maintained to ensure that all requirements are linked to design elements, code, and test cases. This helps in ensuring complete test coverage.
- **Configuration Management:** A robust configuration management system is used to track and control all changes to the software and its related artifacts.

---

11. Software Quality and Its Core Components

Definition of Software Quality

Software quality refers to the degree to which a software product meets specified requirements and user expectations[11]. It is a measure of how well the software is designed and how well it conforms to that design. A high-quality software is one that is reliable, efficient, and easy to use and maintain.

Core Components of Quality

The core components of software quality, often referred to as quality attributes, can be categorized as follows[12]:

- **Functionality:** The ability of the software to perform its specified functions correctly. This includes:
  - **Suitability:** The presence and appropriateness of a set of functions for specified tasks.
  - **Accuracy:** The provision of right or agreed results or effects.
  - **Interoperability:** The ability of the software to interact with other specified systems.
  - **Security:** The ability to prevent unauthorized access and protect data.
- **Reliability:** The ability of the software to perform its required functions under stated conditions for a specified period. This includes:
  - **Maturity:** The frequency of failure due to faults in the software.
  - **Fault Tolerance:** The ability of the software to maintain a specified level of performance in case of software faults or infringements of its specified interface.
  - **Recoverability:** The ability to re-establish its level of performance and recover the data directly affected in case of a failure.
- **Usability:** The ease with which a user can learn, operate, and prepare inputs and interpret outputs through interaction with a system. This includes:
  - **Understandability:** The ease with which the software's functions can be understood.

    - **Learnability:** The effort required to learn how to use the software.
    - **Operability:** The ease with which the software can be operated by the user.
- **Efficiency:** The ability of the software to provide appropriate performance, relative to the amount of resources used under stated conditions. This includes:
    - **Time Behavior:** The speed of response and processing times.
    - **Resource Utilization:** The amount of resources (e.g., CPU, memory) used.
- **Maintainability:** The ease with which a software system or component can be modified to correct faults, improve performance, or adapt to a changed environment. This includes:
    - **Analyzability:** The ease with which deficiencies or causes of failures can be identified.
    - **Changeability:** The ease with which a modification can be implemented.
    - **Stability:** The risk of unexpected effects of modifications.
    - **Testability:** The ease with which modified software can be validated.
- **Portability:** The ability of software to be transferred from one environment to another. This includes:
    - **Adaptability:** The ability of the software to be adapted for different specified environments without applying other actions or means than those provided for this purpose for the software considered.
    - **Installability:** The ease with which the software can be installed in a specified environment.
    - **Co-existence:** The ability of the software to co-exist with other independent software in a common environment sharing common resources.

---

12. Defect Life Cycle

The

Defect Life Cycle, also known as the Bug Life Cycle, is a sequence of stages that a defect goes through from its discovery to its resolution[13].

The stages of the defect life cycle are as follows:

1. **New:** When a tester finds a new defect, it is logged and assigned a "New" status.
2. **Assigned:** The newly logged defect is assigned to a developer or a development team for further analysis.
3. **Open:** The developer starts analyzing the defect and working on a fix.
4. **Fixed:** The developer has fixed the defect and has released the updated code for testing.
5. **Pending Retest:** The defect fix is pending retesting by the testing team.
6. **Retest:** The tester is in the process of retesting the fixed defect.

7. **Verified:** The tester has retested the defect and has confirmed that it is fixed.
8. **Closed:** The defect is officially closed after it has been verified as fixed.
9. **Reopened:** If the tester finds that the defect has not been fixed correctly, they will reopen the defect and it goes back to the "Assigned" or "Open" stage.
10. **Duplicate:** If the defect has been reported earlier or is a duplicate of another existing defect, it is marked as "Duplicate".
11. **Rejected:** If the developer determines that the reported issue is not a valid defect, it is marked as "Rejected". This could be due to a misunderstanding of the requirements or an issue with the test environment.
12. **Deferred:** The defect fix is postponed for a future release. This is usually due to low priority or lack of time.
13. **Not a Bug:** If the reported issue does not impact the functionality of the application, it can be marked as "Not a Bug".