Educational Teaching Material: Software Testing, Verification, Validation & QA

1.5 Software Development Lifecycle (SDLC)

Introduction

The Software Development Lifecycle (SDLC) provides a structured approach to designing, developing, testing, and maintaining software systems. It helps teams plan each step systematically, ensuring the software meets requirements and quality standards.

Definition

- SDLC is a systematic process consisting of distinct phases: planning, requirement analysis, design, development, testing, deployment, and maintenance.

Importance

- Ensures disciplined and predictable software development.
- Reduces project risks, cost, and delivery time.
- Improves software quality and maintainability.
- Provides a clear framework for project management and decision-making.

Relevance to Other Topics

- Integral to understanding testing phases.
- Provides context for quality assurance and quality control practices.
- Establishes foundation for role assignments in a project.

Examples

- Building a mobile banking app following all SDLC phases to ensure security, functionality, and usability.
- Developing university management software using iterative SDLC, allowing feedback incorporation at every stage.

1.6 Roles and Responsibilities of Software Tester, QC, QA, DevOps, and Project Manager

Introduction

Different roles in software development ensure that each aspect of software creation, testing, and delivery is handled efficiently and correctly.

Definitions

- **Software Tester:** Executes test cases, identifies defects, and reports them to the development team.
- **Quality Controller (QC):** Focuses on verifying product functionality and adherence to specifications.
- **Quality Analyst (QA):** Ensures proper processes are in place to prevent defects.
- **DevOps Team:** Bridges development and operations, automates builds, testing, and deployment.
- **Project Manager:** Oversees the entire software project, including planning, execution, monitoring, and delivery.

Importance

- Clarifies responsibility and accountability.
- Ensures quality at all stages of development.
- Facilitates coordination and communication among teams.
- Enhances project efficiency and reduces risk of errors.

Relevance to Other Topics

- Directly linked to SDLC phases.
- Influences testing strategy and execution.
- Essential for proper requirement handling.

Examples

- QA develops the test plan, QC executes the test cases, and a tester logs all defects for an e-commerce app.
- DevOps team sets up continuous integration pipelines for automated deployment and testing.

Unit II: SDLC & Testing (8 Hrs)

2.1 SDLC Stages

Introduction

The SDLC is divided into stages to ensure that software is developed systematically and efficiently.

Definition

- **Planning:** Identify objectives, scope, and resources.
- **Requirement Analysis:** Gather and analyze user needs.
- **Design:** Create software architecture and design specifications.
- **Development:** Write code according to design.
- **Testing:** Verify the software against requirements.
- **Deployment:** Deliver the software to users.
- **Maintenance:** Fix bugs, improve performance, and update features.

Importance

- Ensures clarity and structure in development.
- Reduces time, cost, and risk.
- Supports monitoring and control of the project.

Relevance

- Testing is aligned to SDLC stages.
- Roles of testers and developers are guided by SDLC phase.

Examples

- Requirement gathering for hospital management software.
- Testing phase for fintech application to validate transaction security.

2.2 Software Development Methodologies

Introduction

Methodologies determine the process, workflow, and approach used in software development.

Definition

- **Waterfall:** Sequential development from requirements to maintenance.
- **Agile:** Iterative development with regular feedback and adaptations.
- **Spiral:** Risk-driven iterative model.
- **DevOps:** Emphasis on collaboration, automation, and continuous delivery.

Importance

- Guides team workflow and communication.
- Ensures timely and structured software delivery.
- Balances flexibility and control according to project needs.

Relevance

- Testing strategies and timelines differ across methodologies.
- QA and QC adapt approaches based on methodology.

Examples

- Agile Scrum for incremental mobile app development.
- Waterfall for government projects requiring detailed documentation.

2.3 Types of Testing

Introduction

Testing is essential to verify software functionality, reliability, and performance.

Definition

- **Functional Testing (Dynamic):** Focuses on verifying software functions against requirements.
- **Non-functional Testing:** Assesses attributes such as performance, security, and usability.

Importance

- Ensures the software meets expectations.
- Prevents errors from reaching end users.
- Helps maintain software reliability and quality.

Relevance

- Forms the basis for testing levels and strategies.
- Helps in risk mitigation and defect management.

Examples

- Functional testing of login and payment modules.
- Load testing of online ticket booking system.

2.3.1 Functional (Dynamic) Testing

- Verifies actions and behavior according to specifications.
- Ensures features work correctly under expected scenarios.
- Examples: Testing "Add to Cart" functionality, validating registration workflows.

2.3.2 Non-functional Testing

- Assesses performance, usability, scalability, and security.
- Ensures user satisfaction and system stability.
- Examples: Performance testing with thousands of concurrent users, penetration testing of secure portals.

2.4 Levels of Testing

2.4.1 Unit Testing

- Tests individual code components.

- Detects defects early in development.
- Examples: Testing a tax calculation function, API response verification.

2.4.2 Integration Testing

- Tests combined modules.
- Detects issues at module interfaces.
- Examples: Integrating login with dashboard, combining payment and order modules.

2.4.3 System Testing

- End-to-end testing of complete software.
- Validates overall system behavior.
- Examples: Flight booking workflow testing, ERP system testing.

2.4.4 Acceptance Testing

- Verifies software meets user requirements.
- Conducted before production deployment.
- Examples: UAT for HR management system, alpha testing academic portals.

2.5 Risk of Inadequate Testing

- Leads to defects, failures, and business loss.
- Can affect security, usability, and performance.
- Examples: Banking app errors affecting transactions, healthcare app giving wrong reports.

2.6 Test Platforms

- **Development:** Developers write code and perform unit tests.
- **QC Environment:** Testers perform system and integration testing.
- **UAT Environment:** Clients perform acceptance testing.
- **Production:** Live deployment for real users.
- Examples: Internal QA server for testing, staging server for client demos.

2.7 Software Bug

- Errors causing incorrect results or system failures.
- Examples: Incorrect interest calculation, UI misalignment.

2.7.1 Bug vs Defect

- Bug: Issue detected during testing.
- Defect: Issue reported by user or stakeholder.

• Examples: Tester finds incorrect output (bug), client finds issue post-deployment (defect).

2.7.2 Bug Lifecycle

• Stages: New → Assigned → In Progress → Fixed → Retested → Closed.
• Examples: Login error raised and fixed, report data issue corrected.

2.7.3 Defect Management Process

• Process: Identify → Record → Analyze → Prioritize → Resolve.
• Examples: Jira ticket tracking, severity assignment for payment failure.

Unit III: Requirement Documents (6 Hrs)

3.1 Importance of Requirement Documents

• Ensures clear understanding of software needs.
• Prevents miscommunication and incorrect implementation.
• Examples: Detailed SRS for banking software, BRD for attendance system.

3.1.1 Software Requirement Specification (SRS)

• Specifies functional and non-functional requirements.
• Examples: Login rules, performance constraints.

3.1.2 Business Requirement Document (BRD)

• Captures stakeholder needs and business objectives.
• Examples: E-commerce expectations, CRM business goals.

3.1.3 Functional Requirement Document (FRD)

• Details software operations, inputs, outputs, and workflows.
• Examples: Stepwise workflow specifications, input/output behavior details.