SOFTWARE TESTING AND DEVOPS COURSE

VERSION	LATEST UPDATE	AUTHOR
2.0	28-March 2019	Stanley
2.1	6-June 2019	Stanley
3.0	22-September 2024	Stanley

Course Objectives:

- Understand and apply various software testing methodologies and best practices.
- Learn the principles, tools, and practices of DevOps.
- Bridge the gap between development, testing, and operations teams for continuous delivery and integration.
- Apply automation in testing and DevOps pipelines to optimize the software development lifecycle.

Course Modules Overview

Section 1: Introduction to Software Testing

1. **What is Software Testing?**
    - Importance of testing in the SDLC (Software Development Lifecycle)
    - Testing vs Debugging
    - Manual vs Automated Testing
2. **Types of Testing**:
    - Functional Testing: Unit Testing, Integration Testing, System Testing, UAT
    - Non-Functional Testing: Performance Testing, Load Testing, Security Testing, Usability Testing
    - Regression Testing and Smoke Testing
3. **Software Testing Life Cycle (STLC)**:
    - Requirement Analysis
    - Test Planning
    - Test Case Development
    - Environment Setup
    - Test Execution
    - Test Cycle Closure

Section 2: Testing Techniques and Methodologies

1. **Black Box vs White Box Testing**
2. **Static vs Dynamic Testing**
3. **Exploratory Testing**
4. **Test-Driven Development (TDD)**
    - Writing test cases first, then developing code
5. **Behavior-Driven Development (BDD)**
    - Using tools like Cucumber, Gherkin
6. **Acceptance Test-Driven Development (ATDD)**

Section 3: Test Automation Tools

1. **Introduction to Test Automation**
    - Why automate tests?
    - ROI in test automation
2. **Popular Testing Tools**:
    - **Selenium** (Web Testing)
    - **JUnit** and **TestNG** (Unit Testing)
    - **Cypress** (End-to-End Testing)
    - **Postman** (API Testing)
    - **JMeter** (Performance Testing)
    - **Appium** (Mobile Testing)
3. **Automation Frameworks**:
    - Page Object Model (POM)
    - Data-Driven Testing
    - Keyword-Driven Testing

Section 4: Introduction to DevOps

1. **What is DevOps?**
    - Overview and importance
    - DevOps lifecycle: Continuous Development, Continuous Testing, Continuous Integration (CI), Continuous Deployment (CD), and Continuous Monitoring
    - DevOps vs Agile vs Waterfall
2. **DevOps Culture and Practices**:
    - Collaboration between Development and Operations
    - Infrastructure as Code (IaC)
    - Microservices architecture
3. **The CALMS Framework**: Culture, Automation, Lean, Measurement, Sharing

Section 5: Continuous Integration and Continuous Delivery (CI/CD)

1. **CI/CD Principles**
    - Benefits and Challenges

- Automated builds and testing
- Deployment pipelines

2. **CI/CD Tools**:
   - **Jenkins**, **CircleCI**, **Travis CI**
   - **GitLab CI/CD**
   - **Azure DevOps**
3. **Version Control with Git**:
   - Branching strategies
   - Git workflows for DevOps

Section 6: Public Cloud Platforms vs On-Premise Servers

1. **Introduction to Cloud Computing**:
   - What is cloud computing?
   - The rise of cloud technologies and its impact on DevOps and software testing.
2. **Public Cloud Platforms**:
   - **Key Features**:
     - On-demand scalability
     - Global availability and redundancy
     - Pay-as-you-go pricing
     - Managed services (databases, storage, networking, etc.)
   - **Major Public Cloud Providers**:
     - **Amazon Web Services (AWS)**
     - **Microsoft Azure**
     - **Google Cloud Platform (GCP)**
     - **IBM Cloud** and **Oracle Cloud**
3. **On-Premise Servers**:
   - **Key Features**:
     - Complete control over hardware and software
     - Custom security configurations and compliance
     - Higher upfront costs (CAPEX)
     - Requires dedicated IT infrastructure management
   - **Traditional Use Cases**:
     - Regulated industries (banking, healthcare)

        - Data privacy and residency requirements
        - Legacy system dependencies
4. **Key Differences Between Public Cloud Platforms and On-Premise Servers**:
    - **Cost Model**: CAPEX (on-premise) vs OPEX (cloud)
    - **Scalability**: Instant (cloud) vs Limited/physical (on-premise)
    - **Security**: Shared responsibility model in cloud vs Full control on-premise
    - **Flexibility**: Easy experimentation and provisioning in cloud vs Slow to adapt with on-premise
    - **Management**: Managed services in cloud vs In-house team for on-premise infrastructure
5. **Hybrid Cloud and Multi-Cloud Architectures**:
    - **What is Hybrid Cloud?**
        - A combination of on-premise infrastructure with public or private cloud platforms.
        - Common use cases: Cloud bursting, disaster recovery, and gradual migration.
    - **Multi-Cloud**:
        - Using services from multiple public cloud providers.
        - Benefits: Avoiding vendor lock-in, optimizing performance, and cost efficiency.
6. **DevOps and Testing in Cloud vs On-Premise**:
    - **DevOps in the Cloud**:
        - Cloud-native CI/CD pipelines
        - DevOps automation with infrastructure as code (IaC)
        - Automated scaling and rapid deployment
    - **On-Premise DevOps**:
        - Custom CI/CD pipelines with local resources
        - Hardware limitations and longer deployment cycles
        - Greater focus on maintaining infrastructure
7. **Migration Strategies**:
    - **Lift and Shift**: Moving on-premise workloads directly to the cloud.
    - **Refactoring**: Optimizing applications to take advantage of cloud-native features.
    - **Hybrid Approaches**: Gradual migration while maintaining certain workloads on-premise.
8. **Case Studies and Real-World Scenarios**:
    - Examples of enterprises migrating from on-premise to the cloud.
    - Benefits and challenges experienced during migrations.

9. **Practical Labs:**
    - **Lab 1**: Deploying a sample application on a public cloud platform (AWS/Azure/GCP).
    - **Lab 2**: Comparing cloud infrastructure management (Terraform) with on-premise setup (VMs).
    - **Lab 3**: Configuring a hybrid cloud environment with some services on-premise and others on a public cloud.

Section 7: Infrastructure as Code (IaC) and Configuration Management

1. **Introduction to IaC**:
    - Automating infrastructure setup
    - Provisioning and managing environments (dev, test, prod)
2. **IaC Tools**:
    - **Explore public cloud platforms (AWS, GCP and Azure)**
    - **Terraform**
    - **AWS CloudFormation**
3. **Configuration Management Tools**:
    - **Ansible**
    - **Chef**
    - **Puppet**

Section 8: Containerization and Orchestration

1. **Introduction to Containers**:
    - What are containers and why use them?
    - Benefits over virtual machines (VMs)
2. **Docker**:
    - Basic Docker commands
    - Writing Dockerfiles
    - Docker Compose
3. **Kubernetes**:
    - Introduction to Kubernetes (K8s)
    - Kubernetes architecture (Pods, Services, Nodes)
    - Helm charts for managing K8s configurations

Section 9: Monitoring, Logging, and Incident Management

1. **Application and System Monitoring**:
    - Importance of monitoring in DevOps
    - Key metrics to monitor
2. **Monitoring Tools**:
    - **Prometheus**
    - **Grafana**
    - **Nagios**
    - **Elastic Stack (ELK)**
3. **Logging and Incident Management**:
    - Centralized logging
    - Log aggregation with tools like **Elasticsearch**, **Logstash**, and **Kibana**
    - Incident response and on-call practices

Section 10: Security in DevOps (DevSecOps)

1. **Introduction to DevSecOps**
    - Shifting security to the left
    - Principles of secure code and secure deployments
2. **Security Tools**:
    - **OWASP** for application security
    - **SonarQube** for static code analysis
    - **Snyk** for dependency vulnerability checking
    - Security in CI/CD pipelines

Section 11: Real-World Case Studies and Projects

1. **Case Studies**:
    - CI/CD implementation in enterprises
    - Successful DevOps transformations
2. **Final Project**:
    - Implement a full CI/CD pipeline for a sample application.

    - Write automated tests for an application and integrate them into the pipeline.
    - Set up infrastructure as code and container orchestration for deployment.
    - Configure monitoring and alerting for the deployed application.

Assessment and Evaluation:

- Quizzes and Assignments after each module.
- Mid-term exam to evaluate practical knowledge.
- End of term exam to evaluate theoretical knowledge.
- Final project-based evaluation.

Tools and Technologies Covered:

- **Testing**: Selenium, JUnit/TestNG, Postman, Cypress, JMeter, Appium
- **CI/CD**: Jenkins, GitLab CI, Travis CI, CircleCI, Azure DevOps
- **Version Control**: Git, GitHub/GitLab
- **Infrastructure as Code**: Terraform, Ansible, AWS CloudFormation
- **Containerization**: Docker, Kubernetes, Helm
- **Monitoring**: Prometheus, Grafana, ELK Stack
- **Security**: OWASP, SonarQube, Snyk

Table of Contents

SECTION ONE: INTRODUCTION TO SOFTWARE TESTING

CHAPTER ONE: What is Software Testing?

1.1 Definition

Software Testing is a method used to check whether the actual software product matches expected requirements and to ensure that software product is defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

Some also define Software testing as a White Box and Black Box Testing. In simple terms, Software Testing means the Verification of Application Under Test (AUT).

In section one of this course, you are going to dive deep into developing Software Tests and understand importance of building tests.

1.2 Why Software Testing is Important?

Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Are there examples of where poor Testing resulted into trivial losses?
Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

- Nissan cars recalled over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.

- Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point, the store served coffee for free as they were unable to process the transaction.
- Some of Amazon's third-party retailers saw their product price is reduced to 1% due to a software glitch. They were left with heavy losses.
- China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocents live
- In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.
- In April of 1999, a software bug caused the failure of a $1.2 billion military satellite launch, the costliest accident in history
- In May of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars.

Importance of Testing in the SDLC (Software Development Lifecycle)?

Software testing is a critical phase in the Software Development Lifecycle (SDLC) that ensures the quality, functionality, and reliability of a software product. It serves as the backbone for delivering high-performance applications by detecting defects early, reducing costs, and improving user satisfaction.

Here are the benefits of using software testing:

1. Ensures Quality of the Product

Objective: To validate that the software meets business requirements and customer expectations.

Impact: A well-tested product ensures high quality by eliminating bugs, improving performance, and enhancing the overall user experience. Quality software builds trust with users and minimizes risks.

2. Detects Bugs and Defects Early

Objective: Identify and resolve errors at the earliest stages of development.

Impact: Early bug detection significantly reduces the cost of fixing issues. The later a defect is found in the SDLC, the more costly it becomes to address. By testing early (e.g., through unit tests or integration tests), potential problems are caught before they propagate.

3. Prevents Costly Failures

Objective: Reduce the financial and operational risk of failures.

Impact: Fixing defects in production environments can be incredibly costly in terms of time, money, and resources. Effective testing before deployment mitigates these risks, especially in mission-critical systems, where failures could lead to significant losses.

4. Validates Functionality and Requirements

Objective: Ensure the software meets functional and non-functional requirements.

Impact: Testing verifies whether the software behaves as expected and fulfills all specified requirements (e.g., user authentication, data processing). This validation helps confirm that the software aligns with customer needs and business objectives.

5. Improves Security

Objective: Detect vulnerabilities and security issues within the software.

Impact: Testing helps safeguard against potential threats such as data breaches, unauthorized access, and other cyberattacks. Incorporating security testing (e.g., penetration testing, vulnerability scanning) is crucial for protecting sensitive data and maintaining trust in the software.

6. Enhances Performance and Reliability

Objective: Test the system's performance under different conditions.

Impact: Performance testing ensures that the software is scalable and can handle expected workloads without performance degradation. Load testing and stress testing help uncover performance bottlenecks and allow teams to optimize resource usage.

7. Supports Agile and Continuous Integration (CI) Practices

Objective: Facilitate faster releases without compromising quality.

Impact: In Agile development, frequent testing is necessary to maintain the pace of iterative development. Continuous Integration (CI) pipelines rely on automated testing to catch bugs with every code commit, ensuring continuous delivery of high-quality software.

8. Ensures Usability and User Experience (UX)

Objective: Test from the perspective of the end-user.

Impact: Usability testing ensures that the product is easy to use, intuitive, and delivers a smooth user experience. Focusing on UX testing increases customer satisfaction, which can directly influence product adoption and market success.

9. Facilitates Regulatory Compliance

Objective: Ensure the software complies with legal and industry regulations.

Impact: Some industries (e.g., healthcare, finance) require strict adherence to regulations. Testing helps verify that the product complies with legal standards like GDPR, HIPAA, and PCI-DSS. Compliance testing is essential for avoiding legal penalties.

10. Provides Confidence in the Product's Stability

Objective: Build confidence among stakeholders.

Impact: Thorough testing gives developers, business stakeholders, and customers confidence that the product will perform reliably in real-world scenarios. Stability testing also ensures that changes or updates to the software do not negatively impact other parts of the system.

Difference between Testing vs Debugging

Testing and debugging are two essential but distinct processes in software development. While they both aim to improve software quality, they serve different purposes and are used at different stages of the software lifecycle.

1. Definition:

Testing:

- **Purpose**: Testing is the process of executing a program with the goal of finding defects or validating that the software works as intended.
- **Focus**: It is primarily concerned with identifying bugs, validating features, and ensuring that the software meets functional and non-functional requirements.
- **Output**: The output of testing is a report of any failures, defects, or inconsistencies found.

Debugging:

- **Purpose**: Debugging is the process of identifying, isolating, and fixing the root cause of defects or issues in the code.
- **Focus**: Debugging is about understanding why a program is not working as expected and then resolving the underlying issue.
- **Output**: The output of debugging is the corrected code and confirmation that the issue has been resolved.

2. **Objectives**:

Testing:

- **Detect**: Testing aims to detect bugs, errors, or unexpected behaviors.
- **Verification**: It verifies that the software functions correctly according to the specified requirements.
- **Validation**: Ensures that the product meets the end user's needs (validation).

Debugging:

- **Locate**: Debugging is used to locate the specific part of the code responsible for the failure or bug.
- **Repair**: It aims to fix the underlying code issues and remove errors from the system.
- **Understand**: The developer must understand why the error occurred and prevent it from happening again.

3. Process:

Testing:

- Testing involves executing the software and observing its behavior to ensure it behaves as expected.
- It can be done manually or automatically through different types of testing (unit, integration, system, acceptance).
- Testing usually follows predefined test cases or scripts, covering various use cases and edge cases.

Debugging:

- Debugging is more exploratory and is often triggered after a test fails or a bug is reported.
- The process typically involves using a debugger tool or logging mechanisms to trace the execution flow, inspect variable values, and analyze system behavior.
- Once the issue is identified, the developer makes code modifications and retests the affected areas.

4. Who Performs It?

Testing:

- Typically performed by QA testers or test automation engineers, but developers also conduct unit tests as part of their workflow.
- Testers may not need deep knowledge of the code but focus on the software's external behavior.

Debugging:

- Performed by developers or engineers who wrote or maintain the code.
- Debugging requires a deep understanding of the internal workings of the code and architecture.

5. Timing in the Software Lifecycle:

Testing:

- Testing is a planned activity that happens throughout the SDLC (Software Development Lifecycle).
- It can occur at multiple stages: unit testing during development, integration testing when components interact, system testing before release, and regression testing after changes.

Debugging:

- Debugging is a reactive activity, typically performed after a failure or defect has been detected.
- It is triggered by test failures, bug reports, or unexpected system behavior.

6. Tools Used:

Testing:

- Automation tools: Selenium, JUnit, TestNG, Cypress
- Manual testing tools: TestRail, Postman (for API testing)

- Performance testing tools: JMeter, LoadRunner
- CI/CD tools: Jenkins, CircleCI (for running tests in the pipeline)

Debugging:

- Debuggers: Integrated Debugger in IDEs (like Visual Studio, IntelliJ, Eclipse)
- Log Analysis: Log monitoring tools like ELK (Elasticsearch, Logstash, Kibana), Graylog
- Tracing: Tools like DTrace, OpenTelemetry, and Jaeger for performance tracing

7. Level of Involvement:

Testing:

- Testing can be done without a deep understanding of the code's internal logic.
- It is focused on functional verification, often using black-box testing where testers only look at input/output.

Debugging:

- Debugging requires a deep knowledge of the codebase and how different components interact.
- It involves code-level diagnosis to pinpoint and correct specific faults.

8. **Outcomes**:

Testing:

- The goal of testing is to produce a test report, highlighting defects, failed test cases, and areas that require debugging.
- If the software passes all tests, it can proceed to the next phase of development or deployment.

Debugging:

- The outcome of debugging is fixed code that resolves the specific issue that was causing the test to fail or the system to malfunction.
- After debugging, the affected tests are re-run to verify that the issue is resolved.

Summary:

- Testing is about finding defects and validating functionality, while debugging is about finding the root cause of defects and fixing them.
- Testing is proactive and structured, while debugging is reactive and exploratory.
- Both are essential in ensuring the delivery of high-quality software, with testing focusing on ensuring the software behaves as expected and debugging addressing issues when things go wrong.

CHAPTER TWO: TYPES OF SOFTWARE TESTING

The main categories are **Functional Testing**, **Non-Functional Testing**, **Maintenance Testing**, and testing based on **Automation** and **Manual** methods. Below is a detailed breakdown of the types of software testing within these categories

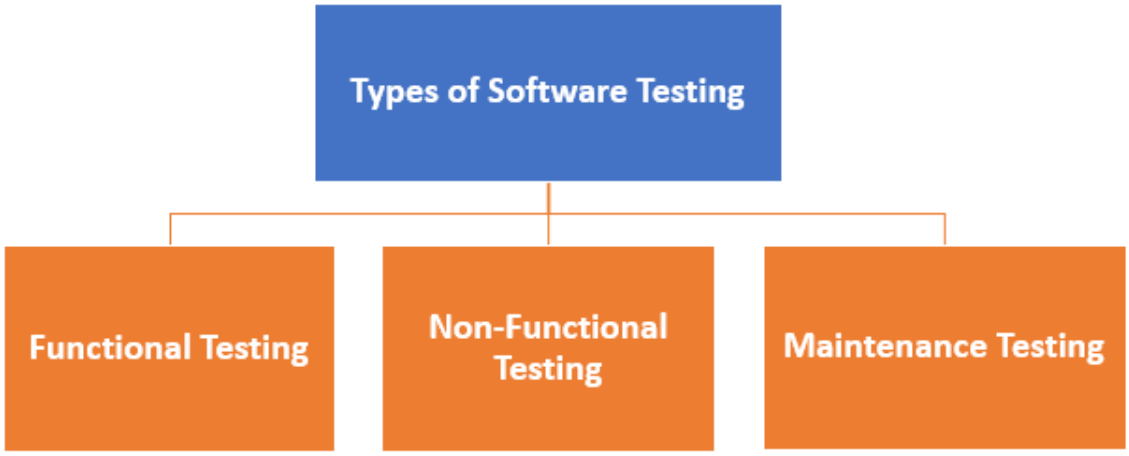


Figure 2.1 Types of software testing

1. Functional Testing

Functional testing focuses on verifying that the software behaves as expected according to the specified requirements. It tests the system's individual features and functions to ensure they work correctly.

- **Types**: Unit testing, integration testing, system testing, and user acceptance testing (UAT).
- **Goal**: To ensure that each function of the software application operates in conformance with the required specification.
- **Example**: Testing login functionality by entering valid and invalid credentials to check if the user is authenticated properly.

2. Non-Functional Testing

Non-functional testing evaluates aspects of the software that do not relate to specific functions but to overall quality attributes like performance, usability, security, and reliability.

- **Types**: Performance testing, load testing, stress testing, security testing, and usability testing.
- **Goal**: To ensure that the software meets certain criteria under load and in various environments.
- **Example**: Running performance tests to check if the software can handle 1,000 simultaneous users without crashing.

3. Maintenance Testing

Maintenance testing occurs after the software has been deployed and is in use. This type of testing ensures that changes such as bug fixes or feature enhancements do not introduce new issues.

- **Types**: Regression testing and retesting.
- **Goal**: To ensure that the application continues to work as expected after updates or modifications.
- **Example**: Testing the system after a patch is deployed to verify that previously working functionalities still operate correctly.

Each of these testing types is important for different phases of the software lifecycle and helps ensure that the final product is reliable and performs as intended.

Testing Category	**Types of Testing**
Functional Testing	• Unit Testing • Integration Testing • Smoke • UAT (User Acceptance Testing) • Localization • Globalization • Interoperability • Etc.
Non-Functional Testing	• Performance • Endurance • Load • Volume • Scalability • Usability • Etc.
Maintenance	• Regression

	• Maintenance

The list of software testing types is endless and not all types are possible to all projects, it depends on each project. The Software testing types can also be further categorized based on execution processes, ie Manual vs Automation as shown in figure 2.2

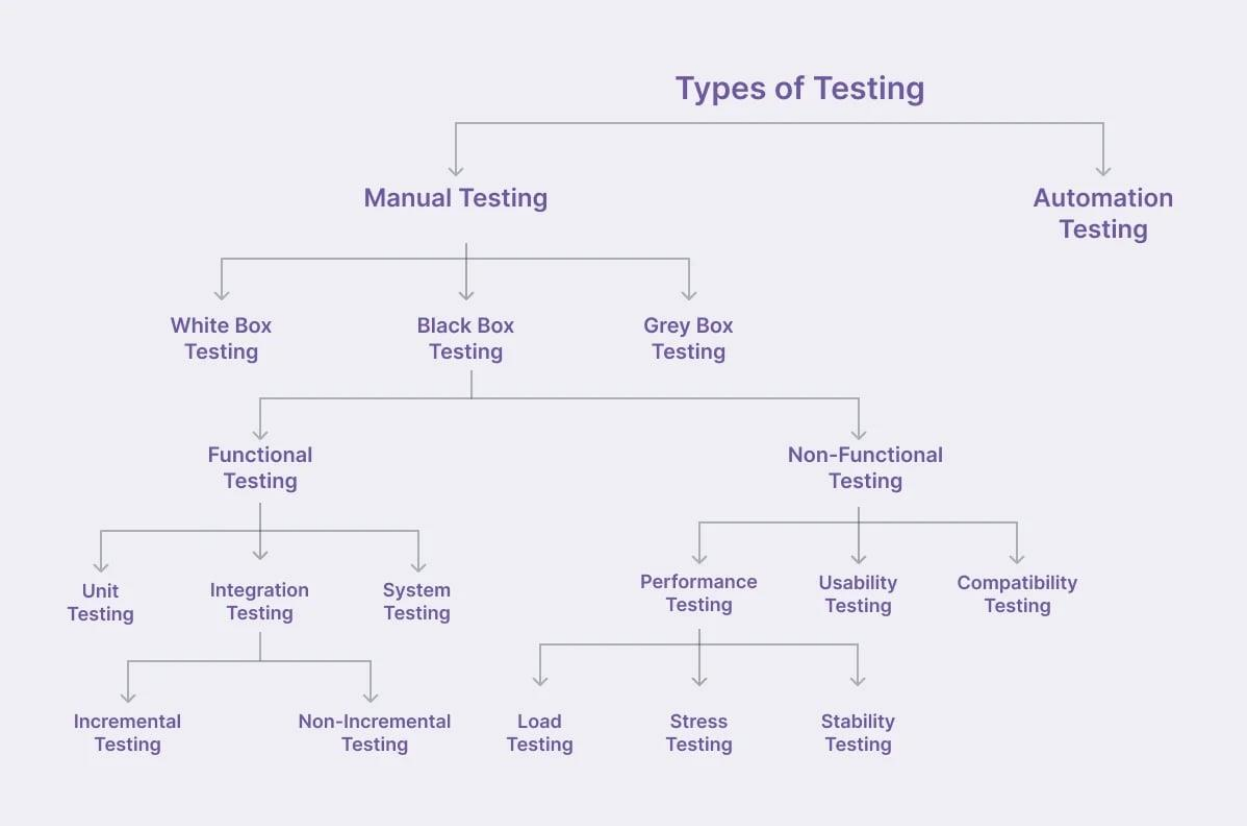


Figure 2.2 Types of software testing execution process

1. White Box Testing

White box testing *techniques analyse the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing.*

White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

Advantages of White box Testing:

- **Thorough Testing** : White box testing is thorough as the entire code and structures are tested.
- **Code Optimization:** It results in the optimization of code removing errors and helps in removing extra lines of code.
- **Early Detection of Defects:** It can start at an earlier stage as it doesn't require any interface as in the case of black box testing.
- **Integration with SDLC:** White box testing can be easily started in the Software Development Life Cycle.
- **Detection of Complex Defects:** Testers can identify defects that cannot be detected through other testing techniques.

2. Black Box Testing

Black-box testing is a type of software testing in which the tester is not concerned with the internal knowledge or implementation details of the software but rather focuses on validating the functionality based on the provided specifications or requirements.

Advantages of Black Box Testing:

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used to find the ambiguity and contradictions in the functional specifications.

3. Gray Box Testing

Gray Box Testing is a software testing technique that is a combination of the Black Box Testing technique and the White Box Testing technique.

1. In the Black Box Testing technique, the tester is unaware of the internal structure of the item being tested and in White Box Testing the internal structure is known to the tester.
2. The internal structure is partially known in Gray Box Testing.
3. This includes access to internal data structures and algorithms to design the test cases.

Advantages of Gray Box Testing:

1. **Clarity of goals:** Users and developers have clear goals while doing testing.
2. **Done from a user perspective:** Gray box testing is mostly done from the user perspective.
3. **High programming skills not required:** Testers are not required to have high programming skills for this testing.
4. **Non-intrusive:** Gray box testing is non-intrusive. The term **non-intrusive** means that the testing approach does not interfere with or alter the system's internal functioning or code during the testing process. Gray box testers typically have partial knowledge of the internal structure of the system but perform tests in a way that mimics a user's interaction with the system, without modifying or deeply penetrating into the system's internals.
5. **Improved product quality:** Overall quality of the product is improved.

Common Types of Black/White Box Testing

1. **Functional Testing**
2. **Non-Functional Testing**

1. Functional Testing

Functional Testing is a type of Software Testing in which the system is tested against the functional requirements and specifications. Functional testing ensures that the requirements or specifications are properly satisfied by the application. This type of testing is particularly concerned with the result of processing. It focuses on the simulation of actual system usage but does not develop any system structure assumptions. The article focuses on discussing function testing.

Benefits of Functional Testing

- **Bug-free product:** Functional testing ensures the delivery of a bug-free and high-quality product.
- **Customer satisfaction:** It ensures that all requirements are met and ensures that the customer is satisfied.

- **Testing focused on specifications:** Functional testing is focused on specifications as per customer usage.
- **Proper working of application:** This ensures that the application works as expected and ensures proper working of all the functionality of the application.
- **Improves quality of the product:** Functional testing ensures the security and safety of the product and improves the quality of the product.

2. Non-Functional Testing

Non-functional Testing is a type of Software Testing that is performed to verify the non-functional requirements of the application. It verifies whether the behavior of the system is as per the requirement or not. It tests all the aspects that are not tested in functional testing. Non-functional testing is a software testing technique that checks the non-functional attributes of the system. It is designed to test the readiness of a system as per non-functional parameters which are never addressed by functional testing. Non-functional testing is as important as functional testing.

Benefits of Non-functional Testing

- **Improved performance:** Non-functional testing checks the performance of the system and determines the performance bottlenecks that can affect the performance.
- **Less time-consuming:** Non-functional testing is overall less time-consuming than the other testing process.
- **Improves user experience:** Non-functional testing like Usability testing checks how easily usable and user-friendly the software is for the users. Thus, focus on improving the overall user experience for the application.
- **More secure product:** As non-functional testing specifically includes security testing that checks the security bottlenecks of the application and how secure is the application against attacks from internal and external sources.

Types of Functional Testing

1. **Unit Testing**
2. **Integration Testing**
3. **System Testing**

1. Unit Testing

Unit testing is a method of testing individual units or components of a software application. It is typically done by developers and is used to ensure that the individual units of the software are working as intended. Unit tests are usually automated and are designed to test specific parts of the code, such as a particular function or method. Unit testing is done at the lowest level of the software development process , where individual units of code are tested in isolation.
Note: Unit Testing basically Included in both White Box Testing and Black Box Testing.

Advantages of Unit Testing:

Some of the advantages of Unit Testing are listed below.

- It helps to identify bugs early in the development process before they become more difficult and expensive to fix.
- It helps to ensure that changes to the code do not introduce new bugs.
- It makes the code more modular and easier to understand and maintain.
- It helps to improve the overall quality and reliability of the software.

Note: *Some popular frameworks and tools that are used for unit testing include* ***JUnit*** *,* ***NUnit,*** *and* ***xUnit.***

- *It's important to keep in mind that Unit Testing is only one aspect of software testing and it should be used in combination with other types of testing such as integration testing, functional testing, and acceptance testing to ensure that the software meets the needs of its users.*
- *It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.*

Example:

1. *In a program we are checking if the loop, method, or function is working fine.*
2. *Misunderstood or incorrect, arithmetic precedence.*
3. *Incorrect initialization.*

2. Integration Testing

Integration testing is a method of testing how different units or components of a software application interact with each other. It is used to identify and resolve any issues that may arise when different units of the software are combined. Integration testing is typically done after unit

testing and before functional testing and is used to verify that the different units of the software work together as intended.

Different Ways of Performing Integration Testing:

Different ways of Integration Testing are discussed below.

- Top-down integration testing: It starts with the highest-level modules and differentiates them from lower-level modules.
- Bottom-up integration testing: It starts with the lowest-level modules and integrates them with higher-level modules.
- Big-Bang integration testing: It combines all the modules and integrates them all at once.
- Incremental integration testing: It integrates the modules in small groups, testing each group as it is added.

Advantages of Integrating Testing

- It helps to identify and resolve issues that may arise when different units of the software are combined.
- It helps to ensure that the different units of the software work together as intended.
- It helps to improve the overall reliability and stability of the software.
- It's important to keep in mind that Integration testing is essential for complex systems where different components are integrated.
- As with unit testing, integration testing is only one aspect of software testing and it should be used in combination with other types of testing such as unit testing, functional testing, and acceptance testing to ensure that the software meets the needs of its users.

The **objective** is to take unit-tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.

Integration testing is of four types: (i) Top-down (ii) Bottom-up (iii) Sandwich (iv) Big-Bang

Example:

1. ***Black Box testing:*** *It is used for validation. In this, we ignore internal working mechanisms and focus on "what is the output?"*
2. ***White box testing:*** *It is used for verification. In this, we focus on internal mechanisms i.e. how the output is achieved.*

3. System Testing

System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users. This type of testing is performed after the integration testing and before the acceptance testing.

System Testing *is a type of software testing that is performed on a completely integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated.*

Advantages of System Testing:

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects that cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real-time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing.

Types of System Testing

1. **Incremental Testing**
2. **Non-Incremental Testing**

1. Incremental Testing

Like development, testing is also a phase of SDLC (Software Development Life Cycle) . Different tests are performed at different stages of the development cycle. Incremental testing is one of the testing approaches that is commonly used in the software field during the testing phase of integration testing which is performed after unit testing . Several stubs and drivers are used to test the modules one after one which helps in discovering errors and defects in the specific modules.

Advantages of Incremental Testing

- Each module has its specific significance. Each one gets a role to play during the testing as they are incremented individually.
- Defects are detected in smaller modules rather than denoting errors and then editing and re-correcting large files.
- It's more flexible and cost-efficient as per requirements and scopes.
- The customer gets the chance to respond to each building.

There are 2 Types of Incremental Testing

1. **Top-down Integration Testing**
2. **Bottom-up Integration Testing**

1. Top-down Integration Testing

Top-down testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving down from top to bottom through the control flow of the architecture structure. In these, high-level modules are tested first, and then low-level modules are tested. Then, finally, integration is done to ensure that the system is working properly. Stubs and drivers are used to carry out this project. This technique is used to increase or stimulate the behavior of Modules that are not integrated into a lower level.

Advantages Top Down Integration Testing

1. There is no need to write drivers.
2. Interface errors are identified at an early stage and fault localization is also easier.
3. Low-level utilities that are not important are not tested well and high-level testers are tested well in an appropriate manner.
4. Representation of test cases is easier and simpler once Input-Output functions are added.

2. Bottom-up Integration Testing

Bottom-up Testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving upward from bottom to top through the control flow of the architecture structure. In these, low-level modules are tested first, and then high-level modules are tested. This type of testing or approach is also known as inductive reasoning and is used as a synthesis synonym in many cases. Bottom-up testing is user-friendly testing and results in an increase in overall software development. This testing results in high success rates with long-lasting results.

Advantages of Bottom-up Integration Testing

- It is easy and simple to create and develop test conditions.
- It is also easy to observe test results.
- It is not necessary to know about the details of the structural design.
- Low-level utilities are also tested well and are also compatible with the object-oriented structure.

Types of Non-functional Testing

1. **Performance Testing**
2. **Usability Testing**
3. **Compatibility Testing**

1. Performance Testing

Performance Testing *is a type of software testing that ensures software applications perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity, and stability under a particular workload.*

Performance testing is a type of software testing that focuses on evaluating the performance and scalability of a system or application. The goal of performance testing is to identify bottlenecks, measure system performance under various loads and conditions, and ensure that the system can handle the expected number of users or transactions.

Advantages of Performance Testing

- Performance testing ensures the speed, load capability, accuracy, and other performances of the system.
- It identifies, monitors, and resolves the issues if anything occurs.
- It ensures the great optimization of the software and also allows many users to use it at the same time.
- It ensures the client as well as the end-customer's satisfaction. Performance testing has several advantages that make it an important aspect of software testing:
- **Identifying bottlenecks** : Performance testing helps identify bottlenecks in the system such as slow database queries, insufficient memory, or network congestion. This helps developers optimize the system and ensure that it can handle the expected number of users or transactions.

2. Usability Testing

You design a product (say a refrigerator) and when it becomes completely ready, you need a potential customer to test it to check it working. To understand whether the machine is ready to come on the market, potential customers test the machines. Likewise, the best example of usability testing is when the software also undergoes various testing processes which is performed by potential users before launching into the market. It is a part of the software development lifecycle (SDLC).

Advantages and Disadvantages of Usability Testing

Usability testing is preferred to evaluate a product or service by testing it with the proper users. In Usability testing, the development and design teams will use to identify issues before coding and the result will be earlier issues will be solved. During a Usability test, you can,

- Learn if participants will be able to complete the specific task completely.
- identify how long it will take to complete the specific task.
- Gives excellent features and functionalities to the product
- Improves user satisfaction and fulfills requirements based on user feedback
- The product becomes more efficient and effective

3. Compatibility Testing

Compatibility testing is software testing that comes under the non functional testing category, and it is performed on an application to check its compatibility (running capability) on different platforms/environments. This testing is done only when the application becomes stable. This means simply this compatibility test aims to check the developed software application functionality on various software, hardware platforms, networks browser etc. This compatibility testing is very important in product production and implementation point of view as it is performed to avoid future issues regarding compatibility.

Advantages of Compatibility Testing

- It ensures complete customer satisfaction.
- It provides service across multiple platforms.
- Identifying bugs during the development process.

There are 4 Types of Performance Testing

1. **Load Testing**
2. **Stress Testing**

3. **Scalability Testing**
4. **Stability Testing**

1. Load Testing

Load testing determines the behavior of the application when multiple users use it at the same time. It is the response of the system measured under varying load conditions.

1. The load testing is carried out for normal and extreme load conditions.
2. Load testing is a type of performance testing that simulates a real-world load on a system or application to see how it performs under stress.
3. The goal of load testing is to identify bottlenecks and determine the maximum number of users or transactions the system can handle.
4. It is an important aspect of software testing as it helps ensure that the system can handle the expected usage levels and identify any potential issues before the system is deployed to production.

Advantages of Load Testing:

Load testing has several advantages that make it an important aspect of software testing:

1. **Identifying bottlenecks:** Load testing helps identify bottlenecks in the system such as slow database queries, insufficient memory, or network congestion. This helps developers optimize the system and ensure that it can handle the expected number of users or transactions.
2. **Improved scalability:** By identifying the system's maximum capacity, load testing helps ensure that the system can handle an increasing number of users or transactions over time. This is particularly important for web-based systems and applications that are expected to handle a high volume of traffic.
3. **Improved reliability:** Load testing helps identify any potential issues that may occur under heavy load conditions, such as increased error rates or slow response times. This helps ensure that the system is reliable and stable when it is deployed to production.

2. Stress Testing

In Stress Testing , we give unfavorable conditions to the system and check how it perform in those conditions.

Example:

1. *Test cases that require maximum memory or other resources are executed.*

2. *Test cases that may cause thrashing in a virtual operating system.*
3. *Test cases that may cause excessive disk requirement Performance Testing.*

It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it, we check, what is the performance of the system in the given load.

Example:

Checking several processor cycles.

3. Scalability Testing

Scalability Testing is a type of non-functional testing in which the performance of a software application, system, network, or process is tested in terms of its capability to scale up or scale down the number of user request load or other such performance attributes. It can be carried out at a hardware, software or database level. Scalability Testing is defined as the ability of a network, system, application, product or a process to perform the function correctly when changes are made in the size or volume of the system to meet a growing need. It ensures that a software product can manage the scheduled increase in user traffic, data volume, transaction counts frequency, and many other things. It tests the system, processes, or database's ability to meet a growing need.

Advantages of Scalability Testing

- It provides more accessibility to the product.
- It detects issues with web page loading and other performance issues.
- It finds and fixes the issues earlier in the product which saves a lot of time.
- It ensures the end-user experience under the specific load. It provides customer satisfaction.
- It helps in effective tool utilization tracking.

4. Stability Testing

Stability Testing *is a type of Software Testing to checks the quality and behavior of the software under different environmental parameters. It is defined as the ability of the product to continue to function over time without failure.*

It is a Non-functional Testing technique that focuses on stressing the software component to the maximum. Stability testing is done to check the efficiency of a developed product beyond normal operational capacity which is known as break point. It has higher significance in error

handling, software reliability, robustness, and scalability of a product under heavy load rather than checking the system behavior under normal circumstances.

Stability testing assesses stability problems. This testing is mainly intended to check whether the application will crash at any point in time or not.

Advantages of Stability Testing

1. It gives the limit of the data that a system can handle practically.
2. It provides confidence on the performance of the system.
3. It determines the stability and robustness of the system under load.
4. Stability testing leads to a better end-user experience.

Other Types of Testing

1. *Smoke Testing*
2. *Sanity Testing*
3. *Regression Testing*
4. *Acceptance Testing*
5. *User Acceptance Testing*
6. *Exploratory Testing*
7. *Adhoc Testing*
8. *Security Testing*
9. *Globalization Testing*
10. *Regression Testing*
11. *Smoke Testing*
12. *Alpha Testing*
13. *Beta Testing*
14. **Object-Oriented Testing**

1. Smoke Testing

Smoke Testing is done to make sure that the software under testing is ready or stable for further testing

It is called a smoke test as the testing of an initial pass is done to check if it did not catch fire or smoke in the initial switch-on.

Example:

If the project has 2 modules so before going to the module make sure that module 1 works properly.

Advantages of Smoke Testing

1. Smoke testing is easy to perform.
2. It helps in identifying defects in the early stages.
3. It improves the quality of the system.
4. Smoke testing reduces the risk of failure.
5. Smoke testing makes progress easier to access.

2. Sanity Testing

It is a **subset** of regression testing . Sanity testing is performed to ensure that the code changes that are made are working properly. Sanity testing is a stoppage to check whether testing for the build can proceed or not. The focus of the team during the sanity testing process is to validate the functionality of the application and not detailed testing. Sanity testing is generally performed on a build where the production deployment is required immediately like a critical bug fix.

Advantages of Sanity Testing

- Sanity testing helps to quickly identify defects in the core functionality.
- It can be carried out in less time as no documentation is required for sanity testing.
- If the defects are found during sanity testing, the project is rejected which is helpful in saving time for execution of regression tests.
- This testing technique is not so expensive when compared to another type of testing.
- It helps to identify the dependent missing objects.

3. Regression Testing

The process of testing the modified parts of the code and the parts that might get affected due to the modifications ensures that no new errors have been introduced in the software after the modifications have been made. Regression means the return of something and in the software field, it refers to the return of a bug.

Advantages of Regression Testing

- It ensures that no new bugs have been introduced after adding new functionalities to the system.

- As most of the test cases used in Regression Testing are selected from the existing test suite, and we already know their expected outputs. Hence, it can be easily automated by the automated tools.
- It helps to maintain the quality of the source code.

4. Acceptance Testing

Acceptance testing is done by the customers to check whether the delivered products perform the desired tasks or not, as stated in the requirements. We use Object-Oriented Testing for discussing test plans and for executing the projects.

Advantages of Acceptance Testing

1. This testing helps the project team to know the further requirements of the users directly as it involves the users for testing.
2. Automated test execution.
3. It brings confidence and satisfaction to the clients as they are directly involved in the testing process.
4. It is easier for the user to describe their requirement.
5. It covers only the Black-Box testing process and hence the entire functionality of the product will be tested.

5. User Acceptance Testing

User Acceptance Testing is a testing methodology where clients/end users participate in product testing to validate the product against their requirements. It is done at the client's site on the developer's site. For industries such as medicine or aerospace, contractual and regulatory compliance testing, and operational acceptance tests are also performed as part of user acceptance tests. UAT is context-dependent and UAT plans are prepared based on requirements and are not required to perform all kinds of user acceptance tests and are even coordinated and contributed by the testing team.

6. Exploratory Testing

Exploratory Testing is a type of software testing in which the tester is free to select any possible methodology to test the software. It is an unscripted approach to software testing. In exploratory testing, software developers use their learning, knowledge, skills, and abilities to test the software developed by themselves. Exploratory testing checks the functionality and

operations of the software as well as identifies the functional and technical faults in it. Exploratory testing aims to optimize and improve the software in every possible way.

Advantages of Exploratory Testing

- **Less preparation required:** It takes no preparation as it is an unscripted testing technique.
- **Finds critical defects:** Exploratory testing involves an investigation process that helps to find critical defects very quickly.
- **Improves productivity:** In exploratory testing, testers use their knowledge, skills, and experience to test the software. It helps to expand the imagination of the testers by executing more test cases, thus enhancing the overall quality of the software.

7. Adhoc Testing

Adhoc testing is a type of software testing that is performed informally and randomly after the formal testing is completed to find any loophole in the system. For this reason, it is also known as Random or Monkey testing. Adhoc testing is not performed in a structured way so it is not based on any methodological approach. That's why Adhoc testing is a type of Unstructured Software Testing.

Advantages of Adhoc testing

- The errors that can not be identified with written test cases can be identified by Adhoc testing.
- It can be performed within a very limited time.
- Helps to create unique test cases.
- This test helps to build a strong product that is less prone to future problems.
- This testing can be performed at any time during Softthe ware Development Life Cycle Process (SDLC)

8. Security Testing

Security Testing is a type of Software Testing that uncovers vulnerabilities in the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss. Security testing of any system is focused on finding all possible loopholes and weaknesses of the system that might result in the loss of information or repute of the organization.

Advantages of Security Testing

1. Identifying vulnerabilities: Security testing helps identify vulnerabilities in the system that could be exploited by attackers, such as weak passwords, unpatched software, and misconfigured systems.
2. Improving system security: Security testing helps improve the overall security of the system by identifying and fixing vulnerabilities and potential threats.
3. Ensuring compliance: Security testing helps ensure that the system meets relevant security standards and regulations, such as HIPAA, PCI DSS, and SOC2.

9. Globalization Testing

Globalization Testing is a type of software testing that is performed to ensure the system or software application can function independently of the geographical and cultural environment. It ensures that the application can be used all over the world and accepts all language texts. Nowadays with the increase in various technologies, every software product is designed in such a way that it is a globalized software product.

Benefits of Globalization Testing

- **Helps to create scalable products:** It makes the software product more flexible and scalable.
- **Save time:** It saves overall time and effort for software testing.
- **Reduce time for localization testing:** Globalization testing helps to reduce the time and cost of localization testing.

10. Regression Testing

Regression testing is a method of testing that is used to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break. It is typically done after changes have been made to the code, such as bug fixes or new features, and is used to verify that the software still works as intended.

Regression testing can be performed in different ways, such as:

- **Retesting** : This involves testing the entire application or specific functionality that was affected by the changes.
- **Re** – **execution** : This involves running a previously executed test suite to ensure that the changes did not break any existing functionality.
- **Comparison** : This involves comparing the current version of the software with a previous version to ensure that the changes did not break any existing functionality.

Advantages of Regression Testing

- It helps to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break.
- It helps to ensure that the software continues to work as intended after changes have been made.
- It helps to improve the overall reliability and stability of the software.
- It's important to keep in mind that regression testing is an ongoing process that should be done throughout the software development
- lifecycle to ensure that the software continues to work as intended. It should be automated as much as possible to save time and resources. Additionally, it's important to have a well-defined regression test suite that covers

Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole component works properly even after adding components to the complete program.

Example:

In school records, suppose we have module staff, students, and finance combining these modules and checking if the integration of these modules works fine in regression testing.

11. Smoke Testing

Smoke Testing is done to make sure that the software under testing is ready or stable for further testing

It is called a smoke test as the testing of an initial pass is done to check if it did not catch fire or smoke in the initial switch-on.

Example:

If the project has 2 modules so before going to the module make sure that module 1 works properly.

12. Alpha Testing

Alpha testing is a type of validation testing. It is a type of acceptance testing that is done before the product is released to customers. It is typically done by QA people.

Example:

When software testing is performed internally within the organisation.

13. Beta Testing

The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment.

Example:

When software testing is performed for the limited number of people.

14. Object-Oriented Testing

Object-Oriented Testing testing is a combination of various testing techniques that help to verify and validate object-oriented software. This testing is done in the following manner:

- Testing of Requirements,
- Design and Analysis of Testing,
- Testing of Code,
- Integration testing,
- System testing,
- User Testing.

Skills required to become a Software Tester

We will discuss the Technical and Non-Technical required to become a Software Tester

Non-Technical Skills

Following skills are essential to become a good Software quality tester. Compare your skill set against the following checklist to determine whether Software Testing is a reality for you-

- **Analytical skills**: A good software tester should have sharp analytical skills. Analytical skills will help break up a complex software system into smaller units to gain a better understanding and create test cases.
- **Communication skill**: A good software tester must have good verbal and written communication skill. Testing artifacts (like test cases/plans, test strategies, bug reports, etc.) created by the software tester should be easy to read and comprehend. Dealing with developers (in the event of bugs or any other issue) will require a shade of discreetness and diplomacy.

- **Time Management & Organization Skills:** Testing at times could be a demanding job especially during the release of code. A software tester must efficiently manage workload, have high productivity, exhibit optimal time management, and organization skills
- **GREAT Attitude:** To be a good software tester you must have a GREAT attitude. An attitude to 'test to break', detail orientation, willingness to learn and suggest process improvements. In the software industry, technologies evolve with an overwhelming speed, and a good software tester should upgrade his/her technical Software testing skills with the changing technologies. Your attitude must reflect a certain degree of independence where you take ownership of the task allocated and complete it without much direct supervision.
- **Passion:** To Excel in any profession or job, one must have a significant degree of the passion for it. A software tester must have a passion for his / her field. BUT how do you determine whether you have a passion for software testing if you have never tested before? Simple TRY it out and if software testing does not excite you switch to something else that holds your interest.

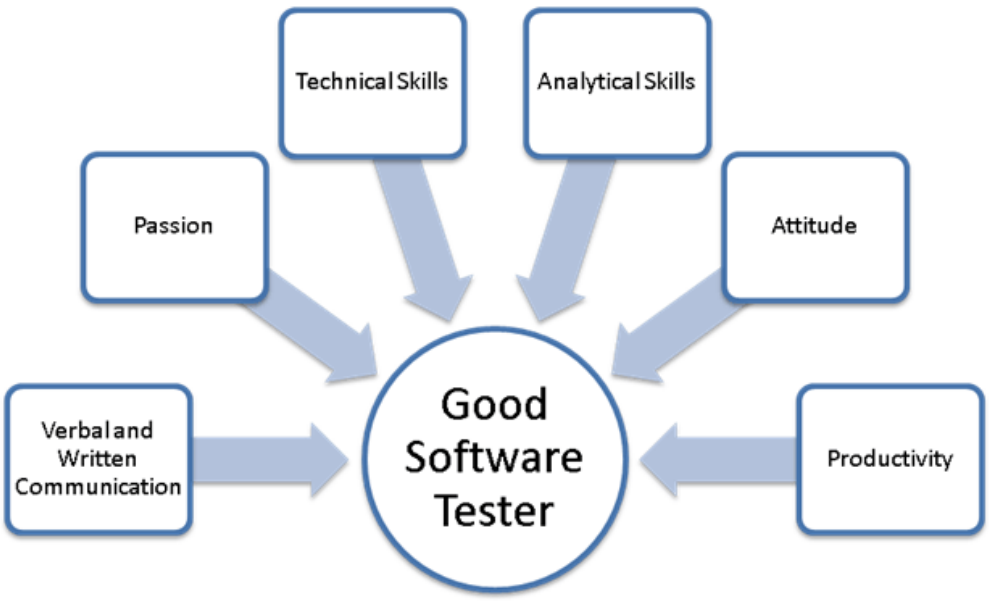


Figure 2.3 Non-Technical skills of a Software Tester

Technical Skills

- **Basic knowledge of Database/ SQL:** Software Systems have a large amount of data in the background. This data is stored in different types of databases like Oracle, MySQL, etc. in the backend. So, there will be situations when this data needs to be validated. In that case, simple/complex SQL queries can be used to check whether proper data is stored in the backend databases.
- **Basic knowledge of Linux commands:** Most of the software applications like Web-Services, Databases, Application Servers are deployed on Linux machines.So it is crucial for testers to have knowledge about Linux commands.
- **Knowledge and hands-on experience of a Test Management Tool:**Test Management is an important aspect of Software testing. Without proper test management techniques, software testing process will fail. Test management is nothing but managing your testing related artifacts.

  For example – A tool like Testlink can be used for tracking all the test cases written by your team.

  There are other tools available that can be utilized for Test Management. So, it is important to have knowledge and working experience of such tools because they are used in most of the companies.

- **Knowledge and hands-on experience of any Defect Tracking tool-** Defect Tracking and Defect life cycle are key aspects of software testing. It is extremely critical to managing defects properly and track them in a systematic manner. Defect tracking becomes necessary because the entire team should know about the defect including managers, developers, and testers. Several tools are used to log defects including QC, Bugzilla, Jira, etc.

- **Knowledge and hands-on experience of Automation tool:** If you see yourself as an "Automation tester" after a couple of years working on manual testing, then you must master a tool and get in-depth, hands-on knowledge of automation tools.

  Knowledge of any scripting language like VBScript, JavaScript, C# is always helpful as a tester if you are looking for a job into automation. Few companies also use Shell/Perl scripting, and there is a lot of demand for testers having knowledge of the same. Again, it will depend on the company and which tools are used by that company.

2.5 Academic Background

Academic background of a software tester should be in Computer Science.

A BTech/ B.E., MCA, BCA (Bachelor of Computer Applications), BSc- Computers, will land you a job quickly.

If you do not hold any of these degrees, then you must complete a software testing certification like ISTQB and CSTE which help you learn Software Development/ Test Life Cycle and other testing methodologies.

2.6 What Does a Software Tester do?

On any typical work day, you will be busy understanding requirement documents, creating test cases, executing test cases, reporting and re-testing bugs, attending review meetings and other team building activities.

2.7 Software Tester Career Path

Your Software Testing career growth as a software tester (QA Analyst) in typical CMMI level 5 company will look like following but will vary from company to company

1. QA Analyst (Fresher)
2. Sr. QA Analyst (2-3 years' experience)

3. QA Team Coordinator (5-6 years' experience)
4. Test Manager (8-11 years' experience)
5. Senior Test Manager (14+ experience)

2.8 How to Become Software Tester

For a complete newbie, here is a common suggested approach to learning Software Testing

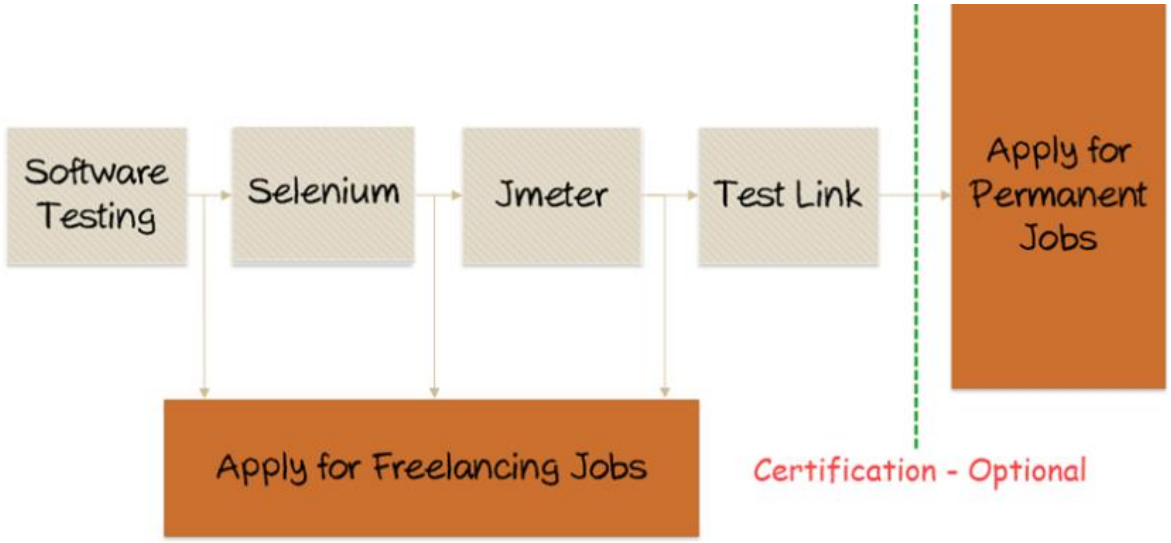


Figure 2.4 Process to become a Software Tester

You start with learning Basic principles of Software Testing. Once done you **apply for freelancing jobs. This will help you gain practical knowledge and will fortify the testing concepts you have learned.**

Next, you proceed to Selenium – Automation tool, then JMeter – Performance Testing tool and finally TestLink – Test Management Tool. All the while you are learning, I recommend you apply for freelancing jobs (apart from other benefits you will earn some income too!).

Once you are through with all the tools, you may consider taking a certification. I recommend ISTQB. However, this is optional.

2.9 Certification Exams:

ISTQB Foundation level is the basic certification in Testing field.

It is not mandatory, but it will help increase your chances of getting the job. Most of the companies have this criterion.

A software tester with ISTQB cleared will be given more priority as compared to others.

After this, when you apply for permanent jobs in big corporations you will have many Software tester skills to offer as well some practical freelancing experience which may be of value and will increase your chances of being selected.

You can also pursue certification in a Testing tool of your choice.

CHAPTER THREE: PRINCIPLES AND STRATEGIES OF SOFTWARE TESTING

3.1: 7 Principles of Software Testing

Background

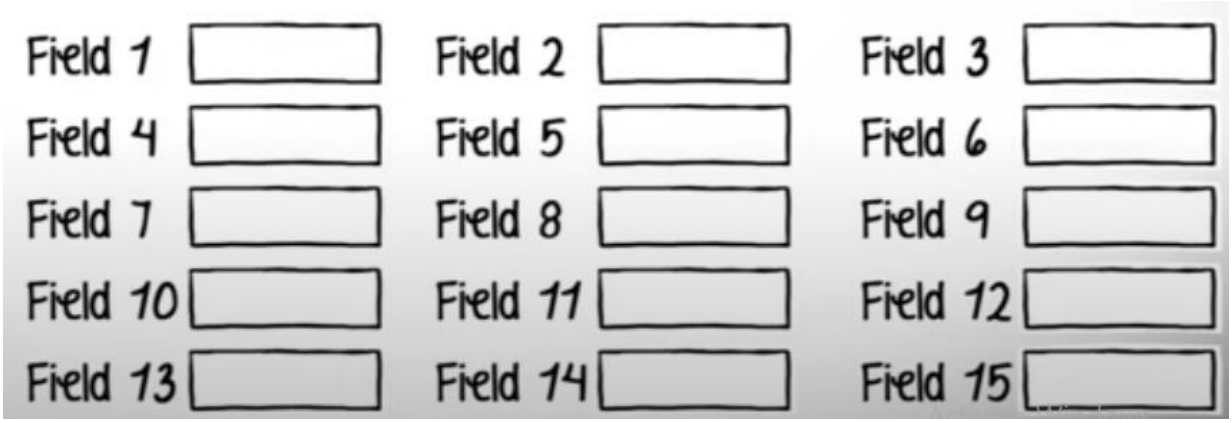
It is important that you achieve optimum test results while conducting software testing without deviating from the goal. However, how you determine that you are following the right strategy for testing? For that, you need to stick to some basic testing principles. Here are the common seven testing principles that are widely practiced in the software industry.

To understand these principles, consider a scenario where you are moving a file from folder A to Folder B.

Think of all the possible ways you can test this.

Apart from the usual scenarios, you can also test the following conditions

- Trying to move the file when it is Open
- You do not have the security rights to paste the file in Folder B
- Folder B is on a shared drive and storage capacity is full.
- Folder B already has a file with the same name, in fact, the list is endless
- Or suppose you have 15 input fields to test, each having 5 possible values, the number of combinations to be tested would be 5^15



If you were to test the entire possible combinations project EXECUTION TIME & COSTS would rise exponentially. We need certain principles and strategies to optimize the testing effort

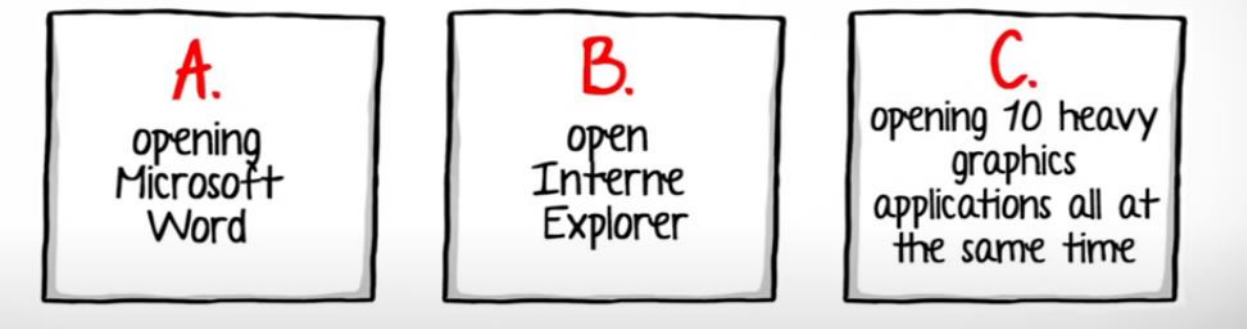
Here are the 7 Principles:

1) Exhaustive testing is not possible

Yes! Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application. **And the million dollar question is, how do you determine this risk?**

To answer this let's do an exercise

In your opinion, Which operation is most likely to cause your Operating system to fail?



It is obviously C.

So if you were testing this Operating system, you would realize that defects are likely to be found in multi-tasking activity and need to be tested thoroughly which brings us to our next principle Defect Clustering

2) Defect Clustering

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle (The *Pareto Principle*, or the 80/20 rule, states that for many phenomena 80% of the result comes from 20% of the effort. The principle

has been named after Vilfredo Pareto—an Italian economist—who, back in 1895, noticed that about 80% of Italy's land belonged to 20% of the country's population.) to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, you can identify such risky modules. But this approach has its own problems

If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

3) Pesticide Paradox

Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects.

To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.

Testers cannot simply depend on existing test techniques. You must look out continually to improve the existing methods to make testing more effective but even after all this sweat & hard work in testing; you can never claim your product is bug-free. You think a company like MICROSOFT would not have tested their OS thoroughly & would risk their reputation just to see their OS crashing during its public launch!

4) Testing shows a presence of defects

Hence, testing principle states that – Testing talks about the presence of defects and don't talk about the absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.

This leads us to our next principle, which states that- Absence of Error is fallacy

5) Absence of Error – fallacy

It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system built is unusable and does not fulfill the user's needs & requirements.

To solve this problem, the next principle of testing states that Early Testing is key

6) Early Testing

Early Testing – Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined.

7) Testing is context dependent

Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software systems are not identical. You might use a different approach, methodologies, techniques, and types of testing depending upon the application type. For instance testing, any POS system at a retail store will be different than testing an ATM machine.

Myth: "Principles are just for reference. I will not use them in practice ."

This is so very untrue. Test Principles will help you create an effective Test Strategy and draft error catching test cases.

But learning testing principles is just like learning to drive for the first time.

Initially, while you learn to drive, you pay attention to each and everything like gear shifts, speed, clutch handling, etc. But with experience, you just focus on driving the rest comes naturally. Such that you even hold conversations with other passengers in the car.

Same is true for testing principles. Experienced testers have internalized these principles to a level that they apply them even without thinking. Hence the myth that the principles are not used in practice is simply not true.

3.2: V-Model in Software Testing

V Model is a highly disciplined SDLC model in which there is a testing phase parallel to each development phase. The V model is an extension of the waterfall model in which testing is done on each stage parallel with development in a sequential way. It is known as the Validation or Verification Model.

Key Software Engineering Terms:

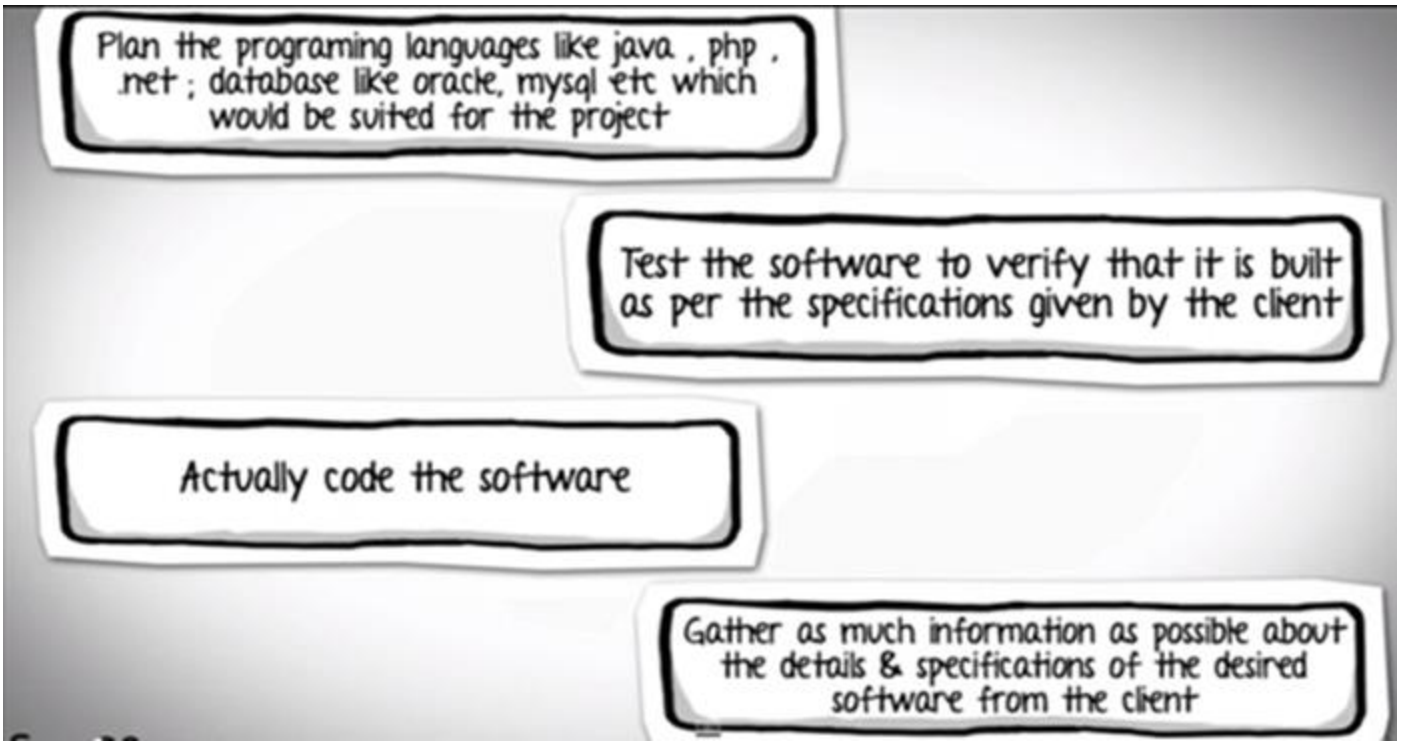
SDLC: SDLC is Software Development Life Cycle. It is the sequence of activities carried out by Developers to design and develop high-quality software.

STLC: STLC is Software Testing Life Cycle. It consists of a series of activities carried out by Testers methodologically to test your software product.

Waterfall Model: Waterfall model is a sequential model divided into different phases of software development activity. Each stage is designed for performing the specific activity. Testing phase in waterfall model starts only after implementation of the system is done.

EXAMPLE To Understand the V Model

Suppose, you are assigned a task, to develop a custom software for a client. Now, irrespective of your technical background, try and make an educated guess about the sequence of steps you will follow, to achieve the task.



The correct sequence would be.

Different phases of the Software Development Cycle	Activities performed in each stage
Requirement Gathering stage	• Gather as much information as possible about the details & specifications of the desired software from the client. This is nothing but the Requirements gathering stage.
Design Stage	• Plan the programming language like Java, PHP, .net; database like Oracle, MySQL, etc. Which would be suited for the project, also some high-level functions & architecture.
Build Stage	• After the design stage, it is build stage, that is nothing but actually code the software

Test Stage	• Next, you test the software to verify that it is built as per the specifications are given by the client.
Deployment stage	• Deploy the application in the respective environment
Maintenance stage	• Once your system is ready to use, you may require to change the code later on as per customer request

All these levels constitute the **waterfall method** of the software development lifecycle.

Problem with the Waterfall Model

As you may observe, that **testing in the model starts only after implementation is done.**

But if you are working in the large project, where the systems are complex, it's easy to miss out the key details in the requirements phase itself. In such cases, an entirely wrong product will be delivered to the client and you might have to start afresh with the project OR if you manage to note the requirements correctly but make serious mistakes in design and architecture of your software you will have to redesign the entire software to correct the error.

Assessments of thousands of projects have shown that **defects introduced during requirements & design make up close to half of the total number of defects.**

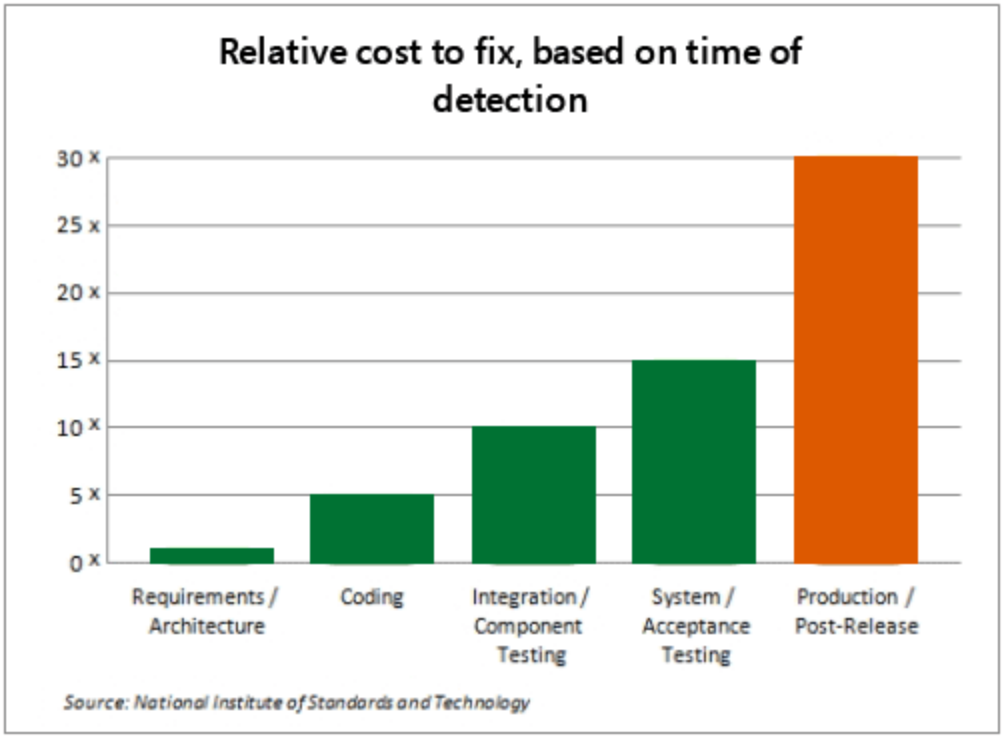


Figure 3.1 cost of fixing bugs

Also, the **costs of fixing a defect increase across the development lifecycle. The earlier in life cycle a defect is detected, the cheaper it is to fix it.** As they say, "A stitch in time saves nine."

Solution: The V Model

To address this concern, **the V model of testing** was developed where **for every phase, in the Development life cycle there is a corresponding Testing phase**

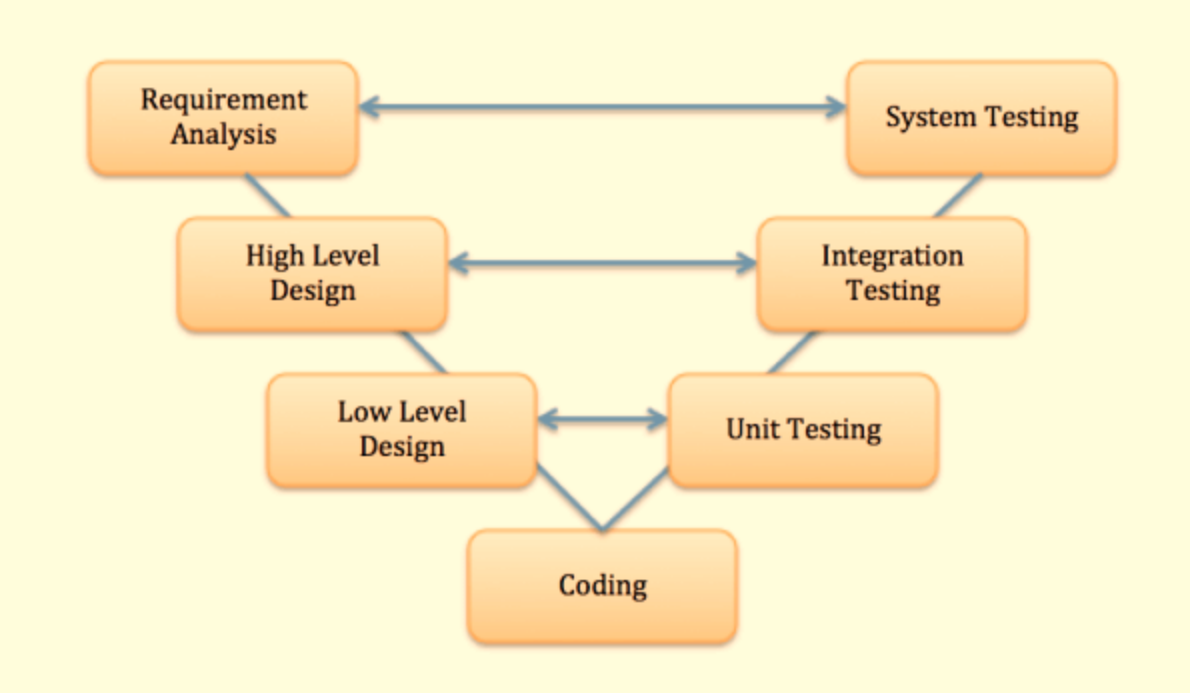


Figure 3.2 V model

- The left side of the model is Software Development Life Cycle – **SDLC**
- The right side of the model is Software Test Life Cycle – **STLC**
- The entire figure looks like a V, hence the name **V – model**

Apart from the V model, there are iterative development models, where development is carried in phases, with each phase adding a functionality to the software. Each phase comprises its independent set of development and testing activities.

Good examples of Development lifecycles following iterative method are Rapid Application Development, Agile Development.

Figure 3.3 illustrates this lifecycle, and highlights the testing activities that take place therein. Each phase of this lifecycle concludes with a verification and validation step, intended to ensure that the deliverable that is produced in the phase is sufficiently trustworthy to serve as a launch pad for the next phase. Strictly speaking, validation ensures that the specification is valid (in the sense that it record all the valid requirements, and nothing but the valid requirements), whereas verification ensures that the product is correct with respect to the specification; hence, in theory,

validation is used at the end of the requirements specification phase, and verification is used in all subsequent phases but in practice, it is a good idea to maintain a healthy suspicion of the specification throughout the lifecycle, to test it at every opportunity and be prepared to adjust it as necessary.

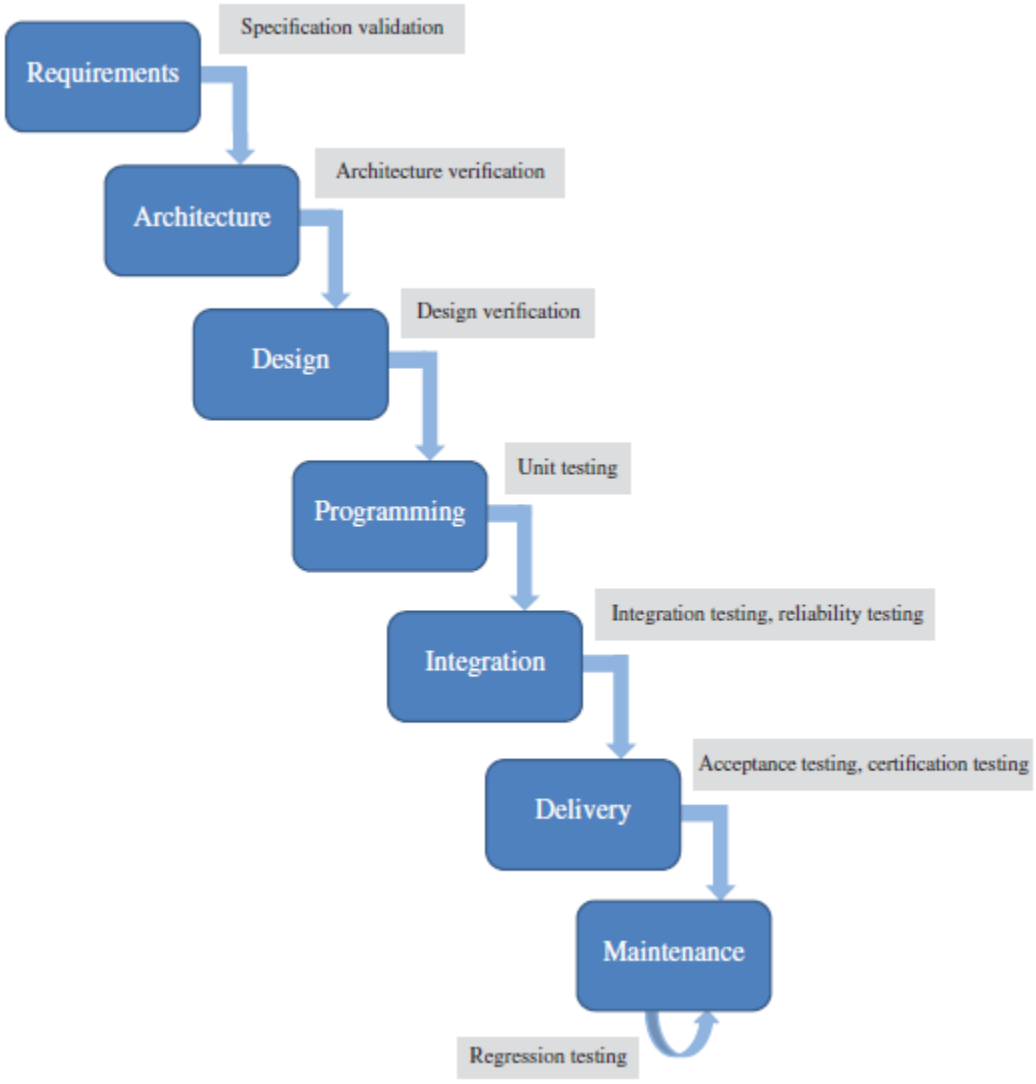


Figure 3.3 SDLC vs STLC

Conclusion

There are numerous development life cycle models. **Development model selected for a project depends on the aims and goals of that project.**

- Testing is not a stand-alone activity, and it has to adapt the development model chosen for the project.
- In any model, testing should be performed at all levels i.e. right from requirements until maintenance.

3.3 STLC (Software Testing Life Cycle) Phases, Entry, Exit Criteria

What is Software Testing Life Cycle (STLC)?

Software Testing Life Cycle (STLC) is a sequence of specific activities conducted during the testing process to ensure software quality goals are met. STLC involves both verification and validation activities. Contrary to popular belief, Software Testing is not just a single/isolate activity, i.e. testing. It consists of a series of activities carried out methodologically to help certify your software product.

STLC Phases

There are following six major phases in every Software Testing Life Cycle Model (STLC Model):

1. Requirement Analysis
2. Test Planning
3. Test case development
4. Test Environment setup
5. Test Execution
6. Test Cycle closure

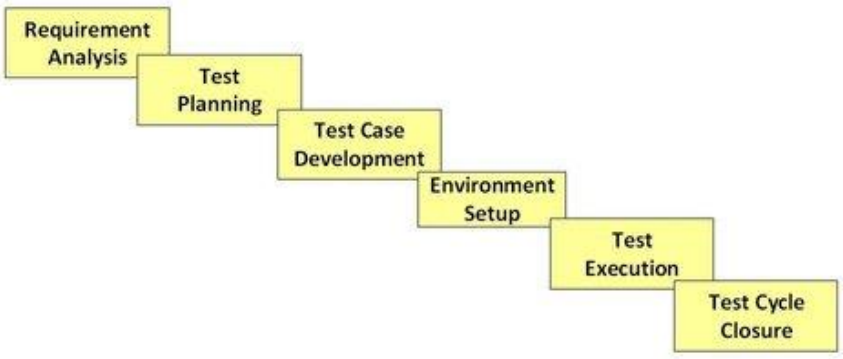


Figure 3.4 STLC Model Phases

Each of these stages has a definite Entry and Exit criteria, Activities & Deliverables associated with it.

What is Entry and Exit Criteria in STLC?

- **Entry Criteria:** Entry Criteria gives the prerequisite items that must be completed before testing can begin.
- **Exit Criteria:** Exit Criteria defines the items that must be completed before testing can be concluded

You have Entry and Exit Criteria for all levels in the Software Testing Life Cycle (STLC)

In an Ideal world, you will not enter the next stage until the exit criteria for the previous stage is met. But practically this is not always possible. So let us focus on activities and deliverables for the different stages in STLC life cycle. Let's look into them in detail.

Requirement Phase Testing

Requirement Phase Testing also known as Requirement Analysis in which test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with various stakeholders to understand requirements in detail. Requirements could be either functional or non-functional. Automation feasibility for the testing project is also done in this stage.

Activities in Requirement Phase Testing

- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM).
- Identify test environment details where testing is supposed to be carried out.
- Automation feasibility analysis (if required).

Deliverables of Requirement Phase Testing

- RTM

- Automation feasibility report. (if applicable)

RTM

The **Requirement Traceability Matrix (RTM)** is a document used in software development to ensure that all project requirements are captured, tracked, and verified throughout the development lifecycle. It establishes a clear relationship between requirements, test cases, and other deliverables. The RTM helps project teams confirm that each requirement has been addressed, tested, and fulfilled, minimizing the risk of missing functionality.

Key Components of an RTM:

1. **Requirement ID**: A unique identifier for each requirement.
2. **Requirement Description**: A detailed explanation of the requirement.
3. **Test Cases**: The corresponding test cases that validate the requirement.
4. **Status**: The status of each requirement (e.g., implemented, tested, pending).
5. **Defects or Issues**: Any bugs or issues related to the requirement.
6. **Priority**: The importance or urgency of the requirement.
7. **Traceability Link**: A reference to other related documents, such as design or development artifacts.

Types of Traceability in RTM:

- **Forward Traceability**: Ensures that all requirements are mapped to their respective test cases and development work.
- **Backward Traceability**: Confirms that the final product's features trace back to the original requirements.
- **Bidirectional Traceability**: Combines both forward and backward traceability, ensuring complete coverage in both directions.

Benefits of RTM:

- **Ensures Coverage**: It ensures all requirements are accounted for and validated.
- **Helps Manage Changes**: Any change in requirements can easily be tracked and linked to test cases or design changes.
- **Improves Project Management**: RTM helps manage scope, track progress, and ensure that no requirement is missed.
- **Reduces Risk**: By documenting the relationship between requirements and testing, RTM reduces the risk of missing critical functionalities.

Examples of RTM

REQUIREMENT TRACEABILITY MATRIX												
Project Name :												
Created By :												
Date :												
Test case IDs → Req. IDs ↓	Req. desc.	Req. type	Test case no. 1	Test case no. 2	Test case no. 3	Test case no. 4	Test case no. 5	Test case no. 6	Test case no. 7	Test case no. 8	Test case no. 9	Test case no. 10
Requirement no. 1			✓						✓			
Requirement no. 2				✓								✓
Requirement no. 3										✓		
Requirement no. 4					✓			✓				✓
Requirement no. 5											✓	
Requirement no. 6						✓						
Requirement no. 7									✓			
Requirement no. 8							✓				✓	
Requirement no. 9										✓		
Requirement no. 10						✓						

REQUIREMENT TRACEABILITY MATRIX							
Project Name:	E-commerce Application						
Project ID:	112						
Business Requirmeent Document (BRD)		Functional Requirements Document (FSD)		Test Case Document			
BR_ ID	BR_ User Case	FR_ID	FR_User Case	Priority	Test Case ID	Status	Comments
BR_1	Product Listing	FR_1	Sort by	High	TC_001 TC_002 TC_004	Finished	Dec 1: Testing started Dec 6: Defect reported Dec 12: Defect Fixed Dec 15: FS_ Passed
		FR_2	Filters	High	TC_001 TC_002 TC_003	Finished	Dec 1: Testing started Dec 6: defect reported Dec 12: Defect Fixed Dec 15: FS_ Passed
BR_2	Payment Module	FR_3	By Credit Card	High	TC_005	In progress	Dec 1: Testing started
		FR_4	By Credit Card	High	TC_006	In progress	Dec 1: Testing started
		FR_5	By Reward/Referral	Medium	TC_007 TC_008	Not Started	

In summary, RTM is an essential tool in project management, ensuring that the software meets all specified requirements and helping to streamline both the development and testing processes.

Test Planning in STLC

Test Planning in STLC is a phase in which a Senior QA manager determines the test plan strategy along with efforts and cost estimates for the project. Moreover, the resources, test environment, test limitations and the testing schedule are also determined. The Test Plan gets prepared and finalized in the same phase.

Test Planning Activities

- Preparation of test plan/strategy document for various types of testing
- Test tool selection
- Test effort estimation
- Resource planning and determining roles and responsibilities.
- Training requirement

Deliverables of Test Planning

- Test plan /strategy document.

- Effort estimation document.

Test Case Development Phase

The **Test Case Development Phase** involves the creation, verification and rework of test cases & test scripts after the test plan is ready. Initially, the Test data is identified then created and reviewed and then reworked based on the preconditions. Then the QA team starts the development process of test cases for individual units.

Test Case Development Activities

- Create test cases, automation scripts (if applicable)
- Review and baseline test cases and scripts
- Create test data (If Test Environment is available)

Deliverables of Test Case Development

- Test cases/scripts
- Test data

Test Environment Setup

Test Environment Setup decides the software and hardware conditions under which a work product is tested. It is one of the critical aspects of the testing process and can be done in parallel with the Test Case Development Phase. Test team may not be involved in this activity if the development team provides the test environment. The test team is required to do a readiness check (smoke testing) of the given environment.

Test Environment Setup Activities

- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data
- Perform smoke test on the build

Deliverables of Test Environment Setup

- Environment ready with test data set up
- Smoke Test Results.

Test Execution Phase

Test Execution Phase is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting. If bugs are reported then it is reverted back to development team for correction and retesting will be performed.

Test Execution Activities

- Execute tests as per plan
- Document test results, and log defects for failed cases
- Map defects to test cases in RTM
- Retest the Defect fixes
- Track the defects to closure

Deliverables of Test Execution

- Completed RTM with the execution status
- Test cases updated with results
- Defect reports

Test Cycle Closure

Test Cycle Closure phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.

Test Cycle Closure Activities

- Evaluate cycle completion criteria based on Time, Test coverage, Cost,Software, Critical Business Objectives, Quality
- Prepare test metrics based on the above parameters.

- Document the learning out of the project
- Prepare Test closure report
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.

Deliverables of Test Cycle Closure

- Test Closure report
- Test metrics

STLC Phases along with Entry and Exit Criteria

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Requirement Analysis	• Requirements Document available (both functional and non functional) • Acceptance criteria defined. • Application architectural document available.	• Analyse business functionality to know the business modules and module specific functionalities. • Identify all transactions in the modules. • Identify all the user profiles. • Gather user interface/ authentication, geographic spread requirements. • Identify types of tests to be performed. • Gather details about testing priorities and focus.	• Signed off RTM • Test automation feasibility report signed off by the client	• RTM • Automation feasibility report (if applicable)

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
		• Prepare Requirement Traceability Matrix (RTM). • Identify test environment details where testing is supposed to be carried out. • Automation feasibility analysis (if required).		
Test Planning	• Requirements Documents • Requirement Traceability matrix. • Test automation feasibility document.	• Analyze various testing approaches available • Finalize on the best-suited approach • Preparation of test plan/strategy document for various types of testing • Test tool selection • Test effort estimation • Resource planning and determining roles and responsibilities.	• Approved test plan/strategy document. • Effort estimation document signed off.	• Test plan/strategy document. • Effort estimation document.

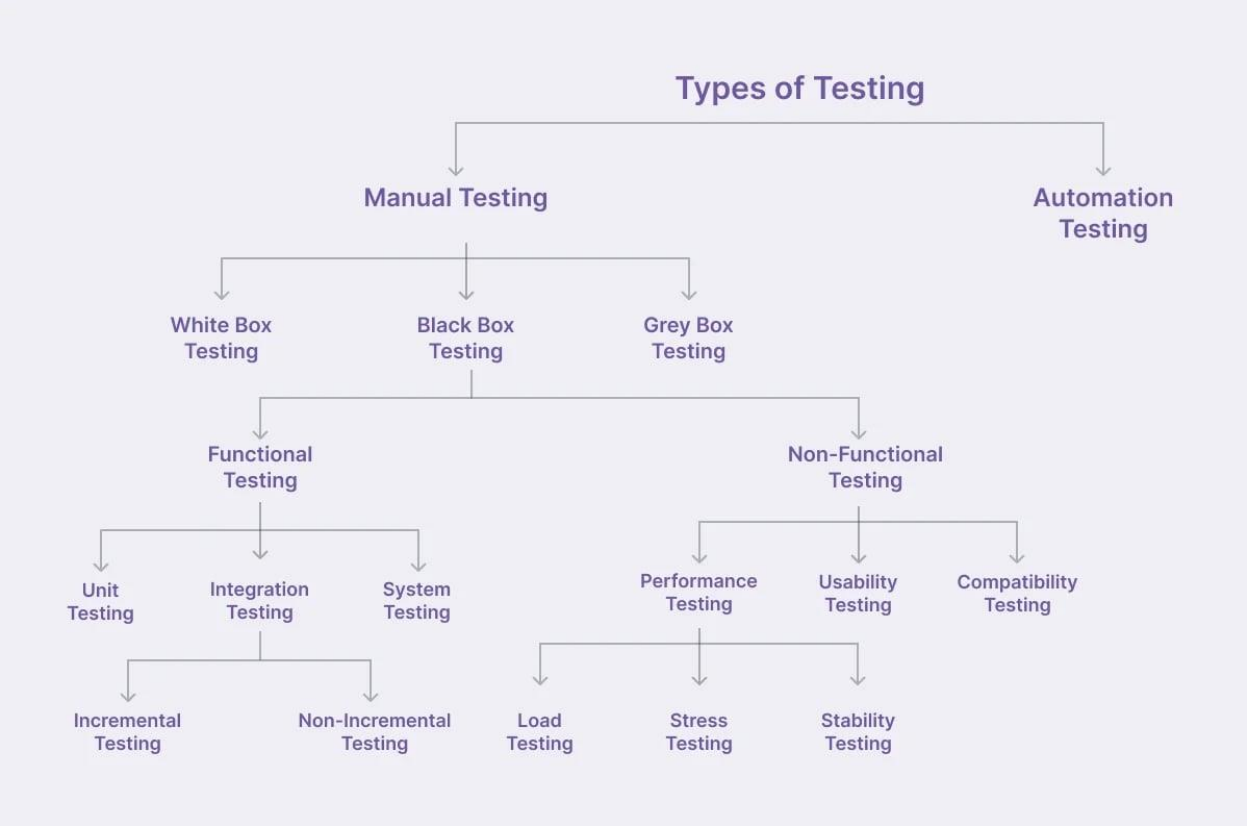
STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Test case development	• Requirements Documents • RTM and test plan • Automation analysis report	• Create test cases, test design, automation scripts (where applicable) • Review and baseline test cases and scripts • Create test data	• Reviewed and signed test Cases/scripts • Reviewed and signed test data	• Test cases/scripts • Test data
Test Environment setup	• System Design and architecture documents are available • Environment set-up plan is available	• Understand the required architecture, environment set-up • Prepare hardware and software development requirement list • Finalize connectivity requirements • Prepare environment setup checklist • Setup test Environment and test data • Perform smoke test on the build • Accept/reject the build depending on smoke test result	• Environment setup is working as per the plan and checklist • Test data setup is complete • Smoke test is successful	• Environment ready with test data set up • Smoke Test Results.

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Test Execution	• Baselined RTM, Test Plan , Test case/scripts are available • Test environment is ready • Test data set up is done • Unit/Integration test report for the build to be tested is available	• Execute tests as per plan • Document test results, and log defects for failed cases • Update test plans/test cases, if necessary • Map defects to test cases in RTM • Retest the defect fixes • Regression Testing of application • Track the defects to closure	• All tests planned are executed • Defects logged and tracked to closure	• Completed RTM with execution status • Test cases updated with results • Defect reports
Test Cycle closure	• Testing has been completed • Test results are available • Defect logs are available	• Evaluate cycle completion criteria based on – Time, Test coverage, Cost, Software Quality, Critical Business Objectives • Prepare test metrics based on the above parameters. • Document the learning out of the project • Prepare Test closure report	• Test Closure report signed	

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
		• Qualitative and quantitative reporting of quality of the work product to the customer. • Test result analysis to find out the defect distribution by type and severity		

CHAPTER FOUR: Manual vs Automated Testing

4.1 Automation Testing Vs. Manual Testing



4.1.1 Manual testing

Manual Testing is a type of software testing in which test cases are executed manually by a tester without using any automated tools. The purpose of Manual Testing is to identify the bugs, issues, and defects in the software application. Manual software testing is the most primitive technique of all testing types and it helps to find critical bugs in the software application. Any new application must be manually tested before its testing can be automated. Manual Software Testing requires more effort but is necessary to check automation feasibility. Manual Testing concepts does not require knowledge of any testing tool. One of the Software Testing Fundamental is "**100% Automation is not possible**". This makes Manual Testing imperative.

Goal of Manual Testing

The key concept of manual testing is to ensure that the application is error free and it is working in conformance to the specified functional requirements.
Test Suites or cases, are designed during the testing phase and should have 100% test coverage.

It also makes sure that reported defects are fixed by developers and re-testing has been performed by testers on the fixed defects.

Basically, this testing checks the quality of the system and delivers bug-free product to the customer.

How to perform Manual Testing

1. Read and understand the software project documentation/guides. Also, study the Application Under Test (AUT) if available.
2. Draft Test cases that cover all the requirements mentioned in the documentation.
3. Review and baseline the test cases with Team Lead, Client (as applicable)
4. Execute the test cases on the AUT
5. Report bugs.
6. Once bugs are fixed, again execute the failing test cases to verify they pass.

Myths of Manual Testing

Following are few common myths and facts related to testing:

Myth: Anyone can do manual testing

Fact: Testing requires many skill sets

Myth: Testing ensures 100% Defect free product

Fact: Testing attempts to find as many defects as possible. Identifying all possible defects is impossible.

Myth: Automated testing is more powerful than manual testing

Fact: 100% test automation cannot be done. Manual Software Testing is also essential.

Myth: Testing is easy

Fact: Testing can be extremely challenging. Testing an application for possible use cases with minimum test cases requires high analytical skills.

Manual Testing Pros and Cons

Pros of Manual Testing:

- Get fast and accurate visual feedback
- It is less expensive as you don't need to spend your budget for the automation tools and process
- Human judgment and intuition always benefit the manual element
- While testing a small change, an automation test would require coding which could be time-consuming. While you could test manually on the fly.

Cons of Manual Testing:

- Less reliable testing method because it's conducted by a human. Therefore, it is always prone to mistakes & errors.
- The manual testing process can't be recorded, so it is not possible to reuse the manual test.
- In this testing method, certain tasks are difficult to perform manually which may require an additional time of the software testing phase.

Tools to Automate Manual Testing

- Selenium
- cypress
- QTP
- Jmeter
- Loadrunner

- TestLink
- Quality Center(ALM)

Conclusion

Manual testing is an activity where the tester needs to be very patient, creative & open minded.

Manual testing is a vital part of user-friendly software development because humans are involved in testing software applications and end-users are also humans. They need to think and act with an End User perspective.

4.1.2 **Automation Testing**

Automation Testing or Test Automation is a software testing technique that performs using special automated testing software tools to execute a test case suite. On the contrary, Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps. The automation testing software can also enter test data into the System Under Test, compare expected and actual results and generate detailed test reports. Software Test Automation demands considerable investments of money and resources.

Successive development cycles will require execution of same test suite repeatedly. Using a test automation tool, it's possible to record this test suite and re-play it as required. Once the test suite is automated, no human intervention is required. The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate Manual Testing altogether.

Why Test Automation?

Test Automation is the best way to increase the effectiveness, test coverage, and execution speed in software testing. Automated software testing is important due to the following reasons:

- Manual Testing of all workflows, all fields, all negative scenarios is time and money consuming
- It is difficult to test for multilingual sites manually
- Test Automation in software testing does not require Human intervention. You can run automated test unattended (overnight)
- Test Automation increases the speed of test execution

- Automation helps increase Test Coverage
- Manual Testing can become boring and hence error-prone.

Which Test Cases to Automate?

Test cases to be automated can be selected using the following criterion to increase the automation ROI

- High Risk – Business Critical test cases
- Test cases that are repeatedly executed
- Test Cases that are very tedious or difficult to perform manually
- Test Cases which are time-consuming

The following category of test cases are not suitable for automation:

- Test Cases that are newly designed and not executed manually at least once
- Test Cases for which the requirements are frequently changing
- Test cases which are executed on an ad-hoc basis.

Automated Testing Process:

Following steps are followed in an Automation Process

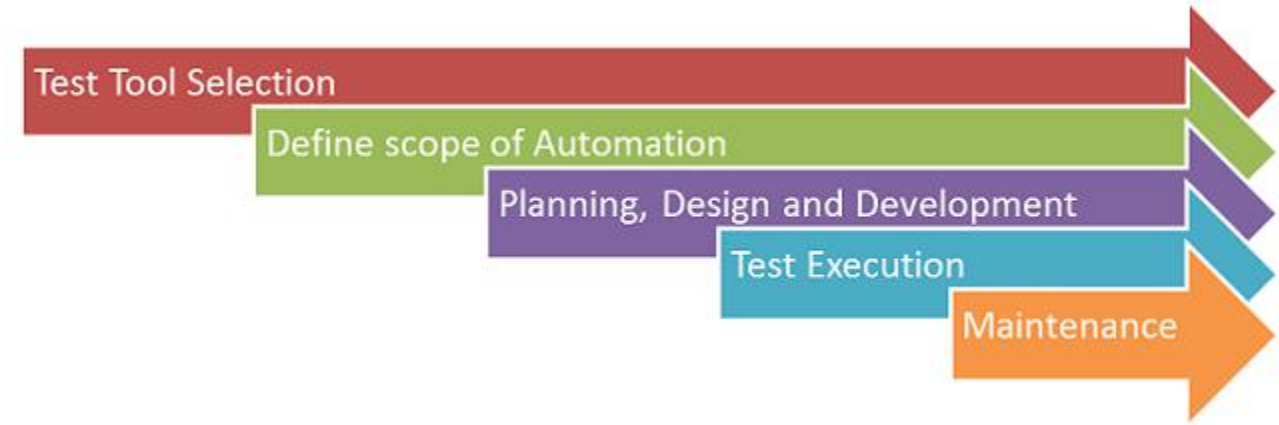
Step 1) Test Tool Selection

Step 2) Define scope of Automation

Step 3) Planning, Design and Development

Step 4) Test Execution

Step 5) Maintenance



Test Automation Process

Test tool selection

Test Tool selection largely depends on the technology the Application Under Test is built on.

Define the scope of Automation

The scope of automation is the area of your Application Under Test which will be automated. Following points help determine scope:

- The features that are important for the business
- Scenarios which have **a large amount of data**
- **Common functionalities** across applications
- Technical feasibility
- The extent to which business components are reused
- **The complexity** of test cases
- Ability to use the same test cases for cross-browser testing

Planning, Design, and Development

During this phase, you create an Automation strategy & plan, which contains the following details-

- Automation tools selected
- Framework design and its features
- In-Scope and Out-of-scope items of automation

- Automation testbed preparation
- Schedule and Timeline of scripting and execution
- Deliverables of Automation Testing

Test Execution

Automation Scripts are executed during this phase. The scripts need input test data before there are set to run. Once executed they provide detailed test reports.

Execution can be performed using the automation tool directly or through the Test Management tool which will invoke the automation tool.

Example: Quality center is the Test Management tool which in turn it will invoke QTP for execution of automation scripts. Scripts can be executed in a single machine or a group of machines. The execution can be done during the night, to save time.

Test Automation Maintenance Approach

Test Automation Maintenance Approach is an automation testing phase carried out to test whether the new functionalities added to the software are working fine or not. Maintenance in automation testing is executed when new automation scripts are added and need to be reviewed and maintained in order to improve the effectiveness of automation scripts with each successive release cycle.

Framework for Automation

It is a conceptual part of the automated testing that helps testers to use resources more efficiently.

A framework is defined as a set of rules or best practices that can be followed in a systematic way that ensures to deliver the desired results.

A framework is set of automation guidelines which help in:

- Maintaining consistency of Testing
- Improves test structuring
- Minimum usage of code

- Less Maintenance of code
- Improve re-usability
- Non-Technical testers can be involved in code
- The training period of using the tool can be reduced
- Involves Data wherever appropriate

These are five types of frameworks used in automation software testing:

Linear Scripting Framework:
This framework is based on the concept of record and playback mode that is always achieved in a linear manner. It is more commonly named as record and playback model.

Typically, in this scripting driven framework, the creation and execution of test scripts is done individually and this framework is an effective way to get started for enterprises.

The automation scripting is done in an incremental manner where every new interaction will be added to the automation tests.

Modular Testing Framework:
Abstraction is the concept on which this framework is built. Based on the modules, independent test scripts are developed to test the software. Specifically, an abstraction layer is built for the components to be hidden from the application under test.

This sort of abstraction concept ensures that changes made to the other part of the application does not affect the underlying components.

Data Driven Testing Framework:
In this testing framework, a separate file in a tabular format is used to store both the input and the expected output results. In this framework, a single driver script can execute all the test cases with multiple sets of data.

This driver script contains navigation that spreads through the program which covers both reading of data files and logging of test status information.

Keyword Driven Testing Framework:

Keyword Driven Testing framework is an application independent framework and uses data tables and keywords to explain the actions to be performed on the application under test. This is more so called as keyword driven test automation framework for web based applications and can be stated as an extension of data driven testing framework.

Hybrid Testing Framework:

This form of hybrid testing framework is the combination of modular, data-driven and keyword test automation frameworks. As this is a hybrid framework, it has been based on the combination of many types of end-to-end testing approaches.

Test Driven Development framework (TDD):

Test driven development is a technique of using automated unit tests to drive the design of software and separates it from any dependencies. Earlier, with traditional testing a successful test could find one or more defects, but by using TDD, it increases the speed of tests and improves the confidence that system meets the requirements and is working properly when compared to traditional testing.

Behavior Driven Development Framework (BDD):

This has been derived from the TDD approach and in this method, tests are more focused and are based on the system behavior. In this approach, the testers can create test cases in simple English language. This simple English language helps even the non-technical people to easily analyze and understand the tests.

Benefits of Automation Testing

Benefits of Automation Testing

Following are the Test Automation benefits:

- 70% faster than the manual testing
- Wider test coverage of application features
- Reliable in results
- Ensure Consistency
- Saves Time and Cost
- Improves accuracy
- Human Intervention is not required while execution
- Increases Efficiency
- Better speed in executing tests
- Re-usable test scripts
- Test Frequently and thoroughly
- More cycle of execution can be achieved through automation
- Early time to market

Automation Testing Tools

There are tons of Functional and Regression Testing Tools available in the market. Here are best test automation tools certified by experts

1) Ranorex Studio

Ranorex Studio is an all-in-one tool for automating functional UI tests, regression tests, data-driven tests and much more. Ranorex Studio includes an easy to use click-and-go interface to automate tests for web, desktop, and mobile applications.

Features:

- Functional UI and end-to-end testing on desktop, web, and mobile
- Cross-browser testing
- SAP, ERP, Delphi and legacy applications.
- iOS and Android
- Run tests locally or remotely, in parallel or distribute on a Selenium Grid
- Robust reporting

2) **Kobiton**

Kobiton's mobile device testing platform offers script-based and scriptless test automation capabilities. Users can create manual tests that can be re-run automatically across a variety of real devices. Kobiton fully supports test automation frameworks such as Appium, Espresso and XCTest, while offering its own scriptless test automation through their NOVA AI.

Features:

- Kobiton's device lab management will let you connect with devices in the cloud, your local on-premises devices as well as on-desk devices.
- Users can automatically create test scripts by converting manual test sessions into scripts that can be executed on multiple devices.
- Easily integrate your defect management system to automatically log tickets with debug sessions attached when a test crashes.

- Kobiton’s Appium Anywhere technology ensures less flaky test scripts by making sure your test runs the same on every device
- Kobiton scriptless test automation generates 100% open standard Appium code for flexible use.

3) ZAPTEST

ZAPTEST is a Software Automation solution for any organization that uses software for operations or development, who is looking to automate it’s back-office operations or software testing processes and to develop a seamless and revolving automation framework.

Features:

- Functional and Performance Test Automation
- RPA (Robotic Process Automation)
- Seamless Test+RPA Software Automation
- Cross-Platform (Parallel) Execution
- DevOps (Mock-ups) Automation
- API Testing
- Auto-Documentation

4) LambdaTest

LambdaTest is one of the most preferred tool to perform automated cross browser testing. They offer an ultra-fast, scalable and secure selenium grid, which users can utilize and run their tests on 2000+ browsers and OS. It supports all latest and legacy browsers.

Features

- Built on latest tech stack, test execution is fast and seamless
- Parallel test execution to shorten test cycles
- Easy integration with various CI/CD execution, project management, team communication tools.

- Users can perform geo location testing and testing of their locally hosted website.
- Users can utilize various APIs to extract all data they need
- Support for all major languages and framework

5) Parasoft Continuous Quality Suite

Parasoft's suite of automated software testing tools integrates quality into the software delivery process for early prevention, detection, and remediation of defects. Parasoft continuous quality suite covers static code analysis, unit testing and code coverage, API testing and service virtualization, and UI testing, enabling delivery at speed and compliance with industry and security standards.

Features:

- AI-powered test creation and execution
- Low-code application testing
- Extensive dashboards for quality reporting and analysis
- Support for 120+ message formats and protocols
- Integration with CI/CD pipeline and Agile DevOps workflows

6) Avo Assure

Avo Assure is a no-code, intelligent, and heterogeneous automation testing solution. With Avo Assure, you can execute test cases without writing a single line of code and achieve more than 90% test automation coverage.

Features:

- 100% no-code
- Heterogeneous – Test across web, windows, non-UI (web services, batch jobs), mobile platforms (Android and IOS), ERPs, Mainframes, and associated emulators
- Accessibility testing
- Smart scheduling to execute test cases in a single VM independently or in parallel. Schedule execution during non-business hours
- Easy-to-understand and intuitive reporting

- 1500+ pre-built keywords and SAP accelerator pack
- Integration with Jira, Jenkins, ALM, QTest, Salesforce, Sauce Labs, TFS, etc

7) Selenium

It is a software testing tool used for Regression Testing. It is an open source testing tool that provides playback and recording facility for Regression Testing. The Selenium IDE only supports Mozilla Firefox web browser.

- It provides the provision to export recorded script in other languages like Java, Ruby, RSpec, Python, C#, etc
- It can be used with frameworks like JUnit and TestNG
- It can execute multiple tests at a time
- Autocomplete for Selenium commands that are common
- Walkthrough tests
- Identifies the element using id, name, X-path, etc.
- Store tests as Ruby Script, HTML, and any other format
- It provides an option to assert the title for every page
- It supports selenium user-extensions.js file
- It allows to insert comments in the middle of the script for better understanding and debugging

8) QTP (MicroFocus UFT)

QTP is widely used for functional and regression testing, it addresses every major software application and environment. To simplify test creation and maintenance, it uses the concept of keyword driven testing. It allows the tester to build test cases directly from the application.

- It is easier to use for a non-technical person to adapt to and create working test cases
- It fix defects faster by thoroughly documenting and replicating defects for developer
- Collapse test creation and test documentation at a single site
- Parameterization is easy than WinRunner
- QTP supports .NET development environment

- It has better object identification mechanism
- It can enhance existing QTP scripts without “Application Under Test” is available, by using the active screen

9) Rational Functional Tester

It is an Object-Oriented automated Functional Testing tool that is capable of performing automated functional, regression, data-driven testing and GUI testing. The main features of this tool are

- It supports a wide range of protocols and applications like Java, HTML, NET, Windows, SAP, Visual Basic, etc.
- It can record and replay the actions on demand
- It integrates well with source control management tools such as Rational Clear Case and Rational Team Concert integration
- It allows developers to create keyword associated script so that it can be re-used
- Eclipse Java Developer Toolkit editor facilitates the team to code test scripts in Java with Eclipse
- It supports custom controls through proxy SDK (Java/.Net)
- It supports version control to enable parallel development of test scripts and concurrent usage by geographically distributed team

10) Watir

It is an open source testing software for regression testing. It enables you to write tests that are easy to read and maintain. Watir supports only internet explorer on windows while Watir webdriver supports Chrome, Firefox, IE, Opera, etc.

- It supports multiple browsers on different platforms
- Rather than using proprietary vendor script, it uses a fully-featured modern scripting language Ruby
- It supports your web app regardless of what it is developed in

11) SilkTest

Silk Test is designed for doing functional and regression testing. For e-business application, silk test is the leading functional testing product. It is a product of Segue Software takeover by Borland in 2006. It is an object-oriented language just like C++. It uses the concept of an object, classes, and inheritance. Its main feature includes

- It consists of all the source script files
- It converts the script commands into GUI commands. On the same machine, commands can be run on a remote or host machine
- To identify the movement of the mouse along with keystrokes, Silktest can be executed. It can avail both playback and record method or descriptive programming methods to get the dialogs
- It identifies all controls and windows of the application under test as objects and determines all of the attributes and properties of each window

KEY DIFFERENCE

- Manual Testing is done manually by QA analyst (Human) whereas Automation Testing is done with the use of script, code and automation tools (computer) by a tester.
- Manual Testing process is not accurate because of the possibilities of human errors whereas the Automation process is reliable because it is code and script based.
- Manual Testing is a time-consuming process whereas Automation Testing is very fast.
- Manual Testing is possible without programming knowledge whereas Automation Testing is not possible without programming knowledge.
- Manual Testing allows random Testing whereas Automation Testing doesn't allow random Testing.

Difference Between Manual Testing and Automation Testing

Parameter	Automation Testing	Manual Testing
Definition	Automation Testing uses automation tools to execute test cases.	In manual testing, test cases are executed by a human tester and software.
Processing time	Automated testing is significantly faster than a manual approach.	Manual testing is time-consuming and takes up human resources.
Exploratory Testing	Automation does not allow random testing	Exploratory testing is possible in Manual Testing
Initial investment	The initial investment in the automated testing is higher. Though the ROI is better in the long run.	The initial investment in the Manual testing is comparatively lower. ROI is lower compared to Automation testing in the long run.
Reliability	Automated testing is a reliable method, as it is performed by tools and scripts. There is no testing Fatigue.	Manual testing is not as accurate because of the possibility of the human errors.
UI Change	For even a trivial change in the UI of the AUT, Automated Test Scripts need to be modified to work as expected	Small changes like change in id, class, etc. of a button wouldn’t thwart execution of a manual tester.
Investment	Investment is required for testing tools as well as automation engineers	Investment is needed for human resources.

Parameter	Automation Testing	Manual Testing
Cost-effective	Not cost effective for low volume regression	Not cost effective for high volume regression.
Test Report Visibility	With automation testing, all stakeholders can login into the automation system and check test execution results	Manual Tests are usually recorded in an Excel or Word, and test results are not readily/ readily available.
Human observation	Automated testing does not involve human consideration. So it can never give assurance of user-friendliness and positive customer experience.	The manual testing method allows human observation, which may be useful to offer user-friendly system.
Performance Testing	Performance Tests like Load Testing, Stress Testing, Spike Testing, etc. have to be tested by an automation tool compulsorily.	Performance Testing is not feasible manually
Parallel Execution	This testing can be executed on different operating platforms in parallel and reduce test execution time.	Manual tests can be executed in parallel but would need to increase your human resource which is expensive
Batch testing	You can Batch multiple Test Scripts for nightly execution.	Manual tests cannot be batched.
Programming knowledge	Programming knowledge is a must in automation testing.	No need for programming in Manual Testing.

Parameter	Automation Testing	Manual Testing
Set up	Automation test requires less complex test execution set up.	Manual testing needs have a more straightforward test execution setup
Engagement	Done by tools. Its accurate and never gets bored!	Repetitive Manual Test Execution can get boring and error-prone.
Ideal approach	Automation testing is useful when frequently executing the same set of test cases	Manual testing proves useful when the test case only needs to run once or twice.
Build Verification Testing	Automation testing is useful for Build Verification Testing (BVT).	Executing the Build Verification Testing (BVT) is very difficult and time-consuming in manual testing.
Deadlines	Automated Tests have zero risks of missing out a pre-decided test.	Manual Testing has a higher risk of missing out the pre-decided test deadline.
Framework	Automation testing uses frameworks like Data Drive, Keyword, Hybrid to accelerate the automation process.	Manual Testing does not use frameworks but may use guidelines, checklists, stringent processes to draft certain test cases.
Documentation	Automated Tests acts as a document provides training value especially for automated unit test cases. A new developer can	Manual Test cases provide no training value

Parameter	Automation Testing	Manual Testing
	look into a unit test cases and understand the code base quickly.	
Test Design	Automated Unit Tests enforce/drive Test Driven Development Design.	Manual Unit Tests do not drive design into the coding process
Devops	Automated Tests help in Build Verification Testing and are an integral part of DevOps Cycle	Manual Testing defeats the automated build principle of DevOps
When to Use?	Automated Testing is suited for Regression Testing, Performance Testing, Load Testing or highly repeatable functional test cases.	Manual Testing is suitable for Exploratory, Usability and Adhoc Testing. It should also be used where the AUT changes frequently.

Automated Testing Pros and Cons

Pros of automated testing:

- Automated testing helps you to find more bugs compare to a human tester
- As most of the part of the testing process is automated, you can have a speedy and efficient process
- Automation process can be recorded. This allows you to reuse and execute the same kind of testing operations
- Automated testing is conducted using software tools, so it works without tiring and fatigue unlike humans in manual testing
- It can easily increase productivity because it provides fast & accurate testing result
- Automated testing support various applications

- Testing coverage can be increased because of automation testing tool never forget to check even the smallest unit

Cons of Automated Testing:

- Without human element, it's difficult to get insight into visual aspects of your UI like colors, font, sizes, contrast or button sizes.
- The tools to run automation testing can be expensive, which may increase the cost of the testing project.
- Automation testing tool is not yet foolproof. Every automation tool has their limitations which reduces the scope of automation.
- Debugging the test script is another major issue in the automated testing. Test maintenance is costly.

4.2 Unit Testing

What is Unit Testing?

UNIT TESTING is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.
In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a WhiteBox testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

4.2.1 Why Unit Testing?

Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost Defect fixing during System Testing, Integration Testing and even Beta Testing after application is built. If

proper unit testing is done in early development, then it saves time and money in the end.
Answers

Here, are the key reasons to perform unit testing in software engineering:

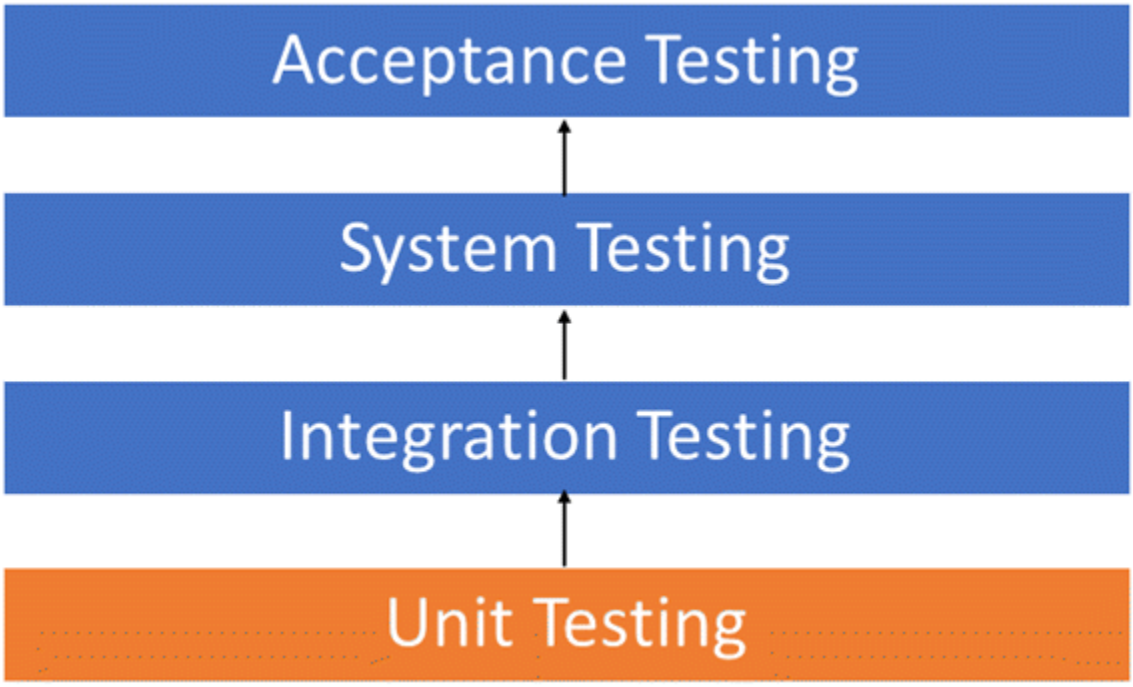


Figure 4.1 Unit Testing Levels

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the testing code base and enables them to make changes quickly
3. Good unit tests serve as project documentation
4. Unit tests help with code re-use. Migrate both your code **and** your tests to your new project. Tweak the code until the tests run again.

4.2.2 How to do Unit Testing

In order **to do Unit Testing**, developers write a section of code to test a specific function in software application. Developers can also isolate this function to test more rigorously which reveals unnecessary dependencies between function being tested and other units so the

dependencies can be eliminated. Developers generally use UnitTest framework to develop automated test cases for unit testing.

Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document.

Under the automated approach-

- A developer writes a section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is deployed.
- A developer could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and paste of code to its own testing environment than its natural environment. **Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces** in the product. These dependencies can then be eliminated.
- A coder generally uses a UnitTest Framework to develop automated test cases. Using an automation framework, the developer codes criteria into the test to verify the correctness of the code. During execution of the test cases, the framework logs failing test cases. Many frameworks will also automatically flag and report, in summary, these failed test cases. Depending on the severity of a failure, the framework may halt subsequent testing.
- **The workflow of Unit Testing is:**

  1) Create Test Cases

  2) Review/Rework

  3) Baseline

  4) Execute Test Cases.

Unit Testing Tools

There are several automated unit test software available to assist with unit testing. I'll provide a few examples below:

1. Junit: Junit is a free to use testing tool used for Java programming language. It provides assertions to identify test method. This tool test data first and then inserted in the piece of code.
2. NUnit: NUnit is widely used unit-testing framework use for all .net languages. It is an open source tool which allows writing scripts manually. It supports data-driven tests

References

1) https://www.guru99.com/software-testing.html
2) https://www.perfecto.io/resources/types-of-testing
3) https://www.softwaretestingclass.com/wp-content/uploads/2016/06/Beginner-Guide-To-Software-Testing.pdf
4) https://videos.gcreddy.com/wp-content/uploads/2019/01/Manual-Testing-Step-by-Step-Tutorial.pdf
5) https://home.cs.colorado.edu/~kena/classes/5828/s12/lectures/05-introtesting.pdf
6) https://www.javatpoint.com/software-testing-tutorial
7) https://www.tutorialspoint.com/software_testing/
8) https://digitalpoint.tech/admin/uploads/4346d933bcfa1d59b368d121f6747980.pdf
9) https://www.imperva.com/learn/application-security/black-box-testing/
10) https://www.udemy.com/course/learn-unit-testing-with-spring-boot/learn/lecture/9968158#overview
11) https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide/learn/lecture/11437138#overview
12)