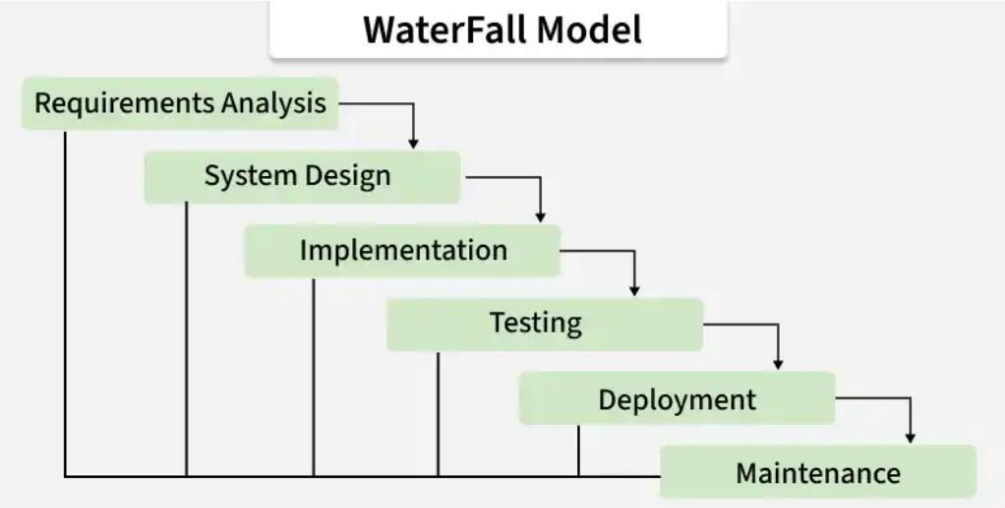
UNIT IV –LIFE CYCLE BASED TESTING

Life Cycle Based Traditional Waterfall Testing, Testing in Iterative Life Cycles, Agile Testing, Agile Model–Driven Development - Model-Based testing: Testing Based on Models - Integration Testing: Decomposition-Based Integration, Call Graph–Based Integration.

4.1 Traditional Waterfall Testing

Waterfall Testing is a sequential testing approach where testing activities are carried out after the software development phase is completed. Each stage follows a fixed order, and the next phase begins only after the previous phase is finished. It is appropriate for projects where the requirements are clearly defined and unlikely to change.

- Testing is performed after the development process is completed.
- Development and testing follow a step-by-step and systematic approach.
- Suitable for small-scale projects with stable and well-defined requirements.



Waterfall Model

Phases of Waterfall Software Testing

The Waterfall model follows a step-by-step sequence, where each phase is completed before the next phase begins. Testing is performed after the implementation phase is completed.

1. Requirement Analysis

The requirements are collected, examined, and documented to identify the needs and expectations of the stakeholders.

2. System Design

The system architecture, design structure, and technical specifications are prepared based on the approved requirements.

3. Implementation (Coding)

Developers convert the design into source code and develop the software according to the specified requirements.

4. Testing

The developed software is systematically tested to detect defects and verify that it satisfies the defined requirements.

5. Deployment

After successful testing, the software is released and made available to end users in the production environment.

6. Maintenance

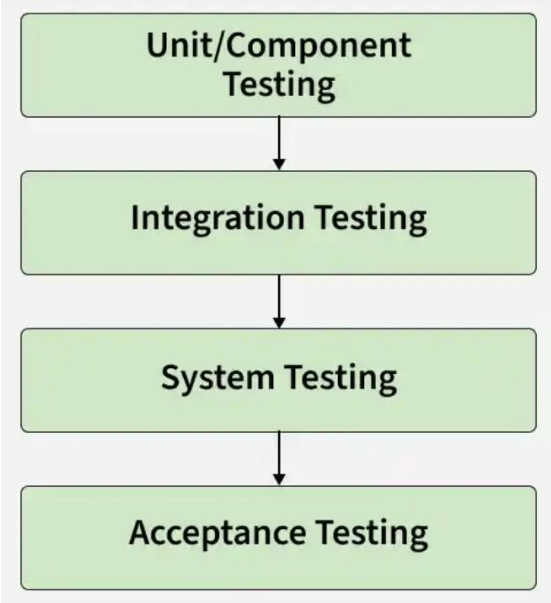
The deployed software is monitored, corrected, and updated to maintain its functionality, performance, and reliability.

Sequential Flow

Requirement Analysis → System Design → Implementation → Testing → Deployment → Maintenance

Testing Levels in Waterfall Testing

Testing is performed in a structured sequence after development:



- **Unit Testing:**
  Each individual module or component is tested independently to ensure that it performs its intended function correctly.
- **Integration Testing:**
  Two or more modules are combined and tested to verify that they communicate, interact, and exchange data correctly.
- **System Testing:**
  The entire integrated software system is tested as a complete product to verify both functional and non-functional requirements.

- **Acceptance Testing:** The completed software is evaluated by customers, clients, or end users to determine whether it satisfies business needs and is suitable for deployment.

Tools Used in Waterfall Testing

Several testing tools can be used in the Waterfall Model to support activities such as test case management, test execution, automation, API validation, and defect tracking. The selection of tools depends on the project type, technology, and testing requirements.

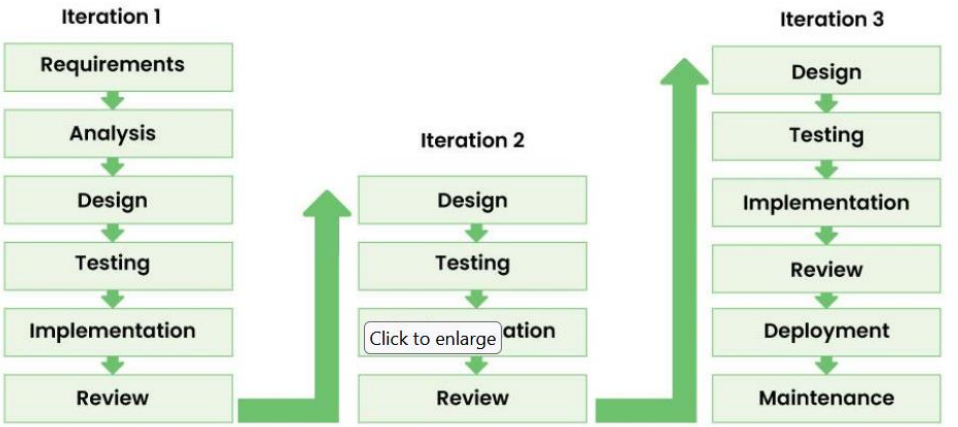
- **Selenium:** Used to automate functional and regression testing of web applications.
- **JUnit / TestNG:** Used to create and execute unit tests for Java-based applications.
- **Postman:** Used to test and validate APIs by sending requests and checking responses.
- **JIRA:** Used to report, track, and manage defects throughout the testing process.
- **HP ALM (Quality Center):** Used to manage requirements, test cases, test execution, and defects.
- **Bugzilla:** Used to record, monitor, and track software defects until they are resolved.

Advantages of Waterfall Testing

The Waterfall Testing approach provides a systematic, organized, and predictable testing process, with clearly defined roles and activities.

- **Clear separation of activities:** Development and testing are handled as separate phases, allowing teams to focus on their specific responsibilities.
- **Structured workflow:** Each phase follows a predefined sequence, making the process easier to understand, plan, and manage.
- **Minimal development interruption:** Since testing begins after coding is completed, developers can concentrate on implementation without frequent testing interruptions.
- **Complete system validation:** The fully developed application can be tested as a whole to identify defects that may occur across the complete system.
- **Suitable for stable requirements:** It works well for projects where requirements are fixed, clearly documented, and unlikely to change**.**

Testing in Iterative Life Cycles



Iterative testing is a testing approach where the product is tested repeatedly in small cycles. After each cycle, the team identifies problems or areas for improvement and makes small changes based on the results. The updated product is then tested again.

Simply, iterative testing means testing, improving, and testing again until the product becomes better and more reliable.

Benefits of Iterative Testing

Iterative testing means testing the software repeatedly in small cycles. After each cycle, errors are identified, improvements are made, and the software is tested again.

1. Early Detection of Errors

Errors can be identified at an early stage instead of waiting until the entire software is completed. This makes fixing the errors easier and less expensive.

Example:
In a Student Management System, the login module is tested first. If the password validation has an error, it can be fixed before developing other modules.

2. Continuous Improvement

The software can be improved after every testing cycle based on the test results and feedback.

Example:
In an online shopping application, the first version may allow users to search for products. After testing, the team may improve the search by adding filters such as price, brand, and rating.

3. Quick Feedback

Testing provides quick feedback to developers about the quality and functionality of the software.

Example:
If users report that the “Add to Cart” button is difficult to find, developers can change its position in the next iteration.

4. Reduced Risk

Since development and testing are performed in small steps, major problems can be avoided. It is easier to manage small changes than a large number of changes at once.

Example:
Instead of developing an entire Banking Application at once, the team can develop and test login, account details, money transfer, and transaction history separately.

5. Better Software Quality

Repeated testing helps identify more defects and gradually improves the reliability, performance, and usability of the software.

Example:
A college attendance system may initially calculate attendance correctly but may become slow when many students use it. Performance testing in a later iteration can identify and fix this problem.

6. Easy to Manage Changes

Requirements may change during development. Iterative testing makes it easier to include new requirements without affecting the entire system.

Example:
If a college decides to add QR-code attendance to an existing attendance system, the new feature can be developed and tested separately.

7. Customer Satisfaction

Customers can provide feedback after each iteration. Their suggestions can be included in the next version of the software.

Example:
A customer using a food delivery application may request a “Live Order Tracking” feature. The development team can add and test this feature in the next iteration.

8. Faster Delivery

A working version of the software can be delivered earlier instead of waiting for the complete system to be finished.

Example:
For a Library Management System, the team can first release a version with book registration and search. Later iterations can add issue/return, fine calculation, and report generation.

Example:

Student Management System

Iteration 1: Develop and test Login → Find errors → Fix errors

Iteration 2: Develop and test Student Registration → Find errors → Fix errors

Iteration 3: Develop and test Mark Entry → Find errors → Fix errors

Iteration 4: Develop and test Reports → Find errors → Fix errors

So, Iterative Testing = Test → Find Problems → Fix → Improve → Test Again.

Agile Testing :
Agile Testing is a software testing approach that follows Agile principles. In this approach, testing is performed continuously along with development activities such as requirement analysis, design, and coding. It is a flexible and iterative method in which testing takes place throughout the SDLC, with emphasis on teamwork, continuous improvement, quick feedback, and customer satisfaction.

- Continuous testing helps identify defects early and supports faster software delivery.
- The entire team participates in testing through effective communication, cooperation, and shared responsibility.
- Regular feedback and simple testing practices help improve software quality and ensure that customer requirements are fulfilled.

Principles of Agile Testing

Agile testing combines **software development and testing activities** throughout the development process. Testing is not postponed until the end. It provides **continuous feedback**, helps find defects early, and ensures that the software meets customer requirements.

1. Continuous Feedback

Feedback is collected regularly during every iteration. This helps the team identify problems early and correct them quickly.

Example: After developing the login feature, testers immediately provide feedback about incorrect password validation.

2. Parallel Testing

Testing is performed **at the same time as development**, rather than waiting until all development is completed.

Example: While developers are creating the registration module, testers can test the completed parts of the module.

3. Team Involvement

Agile testing involves the **entire team**, including developers, testers, business experts, and customers.

Example: A tester reports a problem, the developer fixes it, and the business expert checks whether the solution satisfies the requirement.

4. Lightweight Documentation

Agile teams avoid unnecessary and lengthy documentation. They use **simple test cases, checklists, and user stories** when appropriate.

Example: Instead of preparing a large testing document, the team may maintain a simple checklist such as *Login → Valid username → Valid password → Successful login*.

5. Clean Code

Defects should be corrected as soon as they are identified, preferably within the **same iteration**. This prevents errors from accumulating.

Example: If a calculation error is found during the current sprint, the developer fixes it before the sprint is completed.

6. Regular Feedback

Frequent communication between team members helps the software continuously improve and remain aligned with business requirements.

Example: After each sprint, the team discusses what worked well and what needs to be improved in the next sprint.

7. Customer Focus

Customer requirements are given high priority. Customers can provide feedback and request changes during development.

Example: A customer using an online shopping application requests a **wishlist feature**. The team can consider and add it in a future iteration.

8. Test-Driven Approach

Testing is considered an important part of development. In **Test-Driven Development (TDD)**, tests are created before writing the corresponding code.

Example: First, a test is written to check whether a calculator correctly adds two numbers. Then the developer writes the code to make that test pass.

Agile Testing Strategies:



Phases of Agile Testing

Agile testing is carried out in different phases. Each phase has a specific purpose, from **initial planning to product release and maintenance**.

1. Iteration 0 – Initial Setup

This is the **preparation phase** before actual development begins. The team understands the requirements, identifies possible risks, plans the required resources, and prepares the testing environment.

2. Construction Iteration – Development and Testing

This is the **main development phase**. Features are developed in small iterations, and testing is performed continuously along with development.

Two important types of testing are used:

- **Confirmatory Testing:** Checks whether the developed feature works according to the specified requirements.
- **Investigative Testing:** Explores the software to discover unexpected or hidden defects.

3. Release Phase – Final Testing

In this phase, the completed system undergoes **system testing and acceptance testing**. The goal is to make sure the software is ready for delivery to the customer.

4. Production – Deployment and Maintenance

The software is released to **real users**. After deployment, the team monitors the system and fixes any issues reported by users. Updates and improvements may also be provided.

Agile Testing Quadrants

Agile Testing Quadrants organize different testing activities into four categories. They help the team test both the technical quality and the business/user needs of the software throughout development.

Quadrant 1 (Q1) – Unit and Component Testing

It focuses on testing the internal code and individual components of the software. It mainly includes unit testing, component testing, and integration testing. These tests are usually automated and are mainly performed by developers. The main purpose is to identify coding errors early and ensure that individual parts of the application work correctly.

Example: Testing whether a function correctly calculates a student's total mark.

Quadrant 2 (Q2) – Functional and Business Testing

It focuses on checking whether the software meets business requirements and user expectations. It includes functional testing, user story testing, acceptance testing, and scenario-based testing. Both manual and automated testing can be used.

Example: Testing whether a student can log in, view marks, and download a mark sheet as required.

Quadrant 3 (Q3) – User Experience and Exploratory Testing

It focuses on user experience, usability, and real-world interaction with the application. It includes exploratory testing, usability testing, user acceptance testing, and feedback sessions. These tests are mostly performed manually to understand how easily users can use the application.

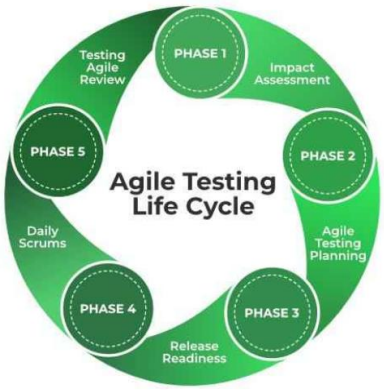
Example: Asking students to use a college portal and checking whether they can easily find the attendance and examination details.

Quadrant 4 (Q4) – Non-Functional Testing

It focuses on the quality characteristics of the software, such as performance, security, scalability, reliability, and stability. These tests are often automated and may be included in CI/CD pipelines for continuous testing.

Example: Testing whether an online examination system can support 1,000 students logging in at the same time without slowing down or crashing.

Life Cycle:



Life Cycle

1. Impact Assessment

Impact assessment involves collecting feedback from users, customers, and stakeholders. The team analyzes this feedback to understand new requirements, identify changes, and decide what needs to be tested in the next iteration.

Example: Users report that the college portal takes too long to load. The team considers this feedback and plans performance testing for the next sprint.

2. Agile Test Planning

In this activity, the team plans all the testing activities for the upcoming sprint. It includes selecting testing methods, tools, resources, schedules, and identifying possible risks.

Example: Before testing a student management system, the team decides which test cases, testing tools, testers, and test environment are required.

3. Release Readiness

Release readiness checks whether the developed features have been properly tested and meet the required quality standards. The team decides whether the software is ready for release or needs further improvement.

Example: Before releasing an online examination system, the team checks whether all major defects are fixed and whether the system performs correctly.

4. Daily Scrums

Daily Scrum is a short daily meeting where team members discuss their progress, current testing activities, problems, and plans for the day. It helps everyone stay coordinated.

Example: A tester may say, “Yesterday I completed login testing. Today I will test student registration. I am currently facing an issue with the test environment.”

5. Test Agility Review

A Test Agility Review is conducted regularly to evaluate the testing process and its results. The team discusses what went well, what problems occurred, and what improvements should be made in the next testing cycle.

Example: After completing a sprint, the team finds that some defects were reported late. They decide to involve testers earlier in the next sprint.