Week 1:

Day 1-2:

Block 1: Introduction to QA

- [x] ~~Overview of Quality Assurance~~
- [x] ~~Importance of QA in software development~~

Block 2: Types of Testing

- [x] ~~**“Understanding manual and automated testing.”**~~

1. Manual Testing:

Involves testers executing test cases without using any automation tools. Testers simulate end-user scenarios to ensure the application behaves as expected.

Advantages:

- Effective for exploratory testing.
- Suitable for small-scale and one-time projects.
- Useful for usability testing and ad-hoc testing.

Disadvantages:

- Time-consuming for repetitive tasks.
- Prone to human errors.
- Not feasible for large-scale or frequent testing.

2. Automated Testing:

Automated testing uses tools and scripts to execute predefined test cases. It is particularly effective for repetitive and regression testing.

Advantages:

- Faster execution of test cases.
- Reusable test scripts for regression testing.
- Suitable for large-scale and repetitive tasks.

Disadvantages:

- Initial setup and scripting can be time-consuming.
- Not as effective for usability and exploratory testing.
- Requires maintenance as the application evolves.

- [x] ~~**“Overview of functional, non-functional, and regression testing.”**~~

Functional Testing:

1. Unit Testing:

- Scope: Individual units or components of the software.
- Purpose: Validate that each unit performs as designed.

2. Integration Testing:

- Scope: Testing the combination of units or systems.
- Purpose: Ensure that integrated components work together correctly.

3. System Testing:
   - Scope: Entire system or software.
   - Purpose: Verify that the entire system meets specified requirements.
4. Acceptance Testing:
   - Scope: Validates if the system meets user requirements.
   - Purpose: Determines if the system is ready for release.
   - Non-Functional Testing:

1. Performance Testing:
   - Types: Load testing, stress testing, scalability testing.
   - Purpose: Evaluate system performance under different conditions.
2. Security Testing:
   - Purpose: Identify vulnerabilities in the system.
   - Activities: Penetration testing, vulnerability scanning.
3. Usability Testing:
   - Purpose: Evaluate the user-friendliness of the system.
   - Activities: User interface testing, user experience testing.
4. Reliability and Maintainability Testing:
   - Purpose: Assess the system's reliability and ease of maintenance.
   - Regression Testing:
   - Purpose: Ensure that new code changes do not adversely affect existing functionalities.
   - Execution: Often automated, involving the re-execution of previously executed test cases.

Day 3-4:

Block 3: QA Life Cycle

- [x] ~~**“ Phases of the QA life cycle”**~~

1. Requirement Analysis:
   - Objective: Understand and analyze the project requirements.
   - Activities: QA team collaborates with stakeholders to gather and clarify requirements.
2. Test Planning:
   - Objective: Plan the QA strategy and define testing objectives.
   - Activities: Develop a test plan, including scope, resources, schedule, and test cases.
3. Test Case Design:
   - Objective: Create detailed test cases based on requirements.
   - Activities: Identify test scenarios, design test cases, and create test data.
4. Environment Setup:
   - Objective: Set up the testing environment.
   - Activities: Configure hardware, software, and network environments for testing.
5. Test Execution:
   - Objective: Execute test cases and report defects.
   - Activities: Run test cases, record results, and report any identified issues.
6. Defect Tracking:
   - Objective: Document and manage defects.

- Activities: Log defects, prioritize them, and track their resolution.

7. Regression Testing:
    - Objective: Ensure that new changes do not adversely impact existing functionality.
    - Activities: Re-run previously executed test cases to verify system stability.
8. Release and Deployment:
    - Objective: Approve the software for release.
    - Activities: Evaluate the overall quality and readiness of the product for release.
9. Post-Release Monitoring:
    - Objective: Monitor the software in a live environment.
    - Activities: Gather feedback, address post-release issues, and plan for future improvements.

- [x] **~~“Importance of early testing.”~~**

1. Early Identification of Defects:
    - Benefits: Early testing helps catch defects in the initial stages of development when they are less costly to fix.
2. Cost Savings:
    - Benefits: Fixing defects early in the development process is more cost-effective than addressing them later or after the product release.
3. Improved Product Quality:
    - Benefits: Early testing contributes to a higher-quality product by identifying and resolving issues promptly.
4. Reduced Time to Market:
    - Benefits: Early testing accelerates the development process by preventing the accumulation of major defects, leading to faster time-to-market.
5. Enhanced Stakeholder Confidence:
    - Benefits: Early testing instills confidence in stakeholders by demonstrating the project's commitment to quality.
6. Optimized Resource Utilization:
    - Benefits: Early testing optimizes the use of resources by avoiding rework and minimizing disruptions to the development workflow.

Block 4: Testing Documentation

- [x] **~~“Introduction to test plans, test cases, and test scripts.”~~**

Test Plan:
A test plan is a comprehensive document that outlines the approach, scope, resources, schedule, and activities planned for testing a specific software product or system.

Key Components:

- Test Case ID:
    - A unique identifier for each test case.
- Test Case Description:

    - Clear and concise description of the specific test scenario.
- Test Steps:
    - Detailed step-by-step instructions to execute the test.
- Test Data:
    - Input data required for the test.
- Expected Result:
    - The anticipated outcome if the system behaves as expected.
- Actual Result:
    - The observed result after executing the test.
- Preconditions:
    - Any required conditions that must be met before the test can be executed.
- Postconditions:
    - The state of the system after the test is executed.

Test Scripts:
A test script is a set of instructions written in a programming language that is used to automate the execution of test cases.

Key Components:
- Script Header:
    - Information about the script, such as its purpose, author, and creation date.
- Test Case Reference:
    - Links to the specific test case being automated.
- Test Data Setup:
    - Instructions for preparing the test environment and input data.
- Test Execution Steps:
    - Automated steps to simulate user interactions with the software.
- Assertions:
    - Checks and validations to verify the expected outcomes.
- Cleanup Steps:
    - Instructions for restoring the system to its original state after the test.
- Logging and Reporting:
    - Logging of relevant information during script execution and generating test reports.

- [x] ~~**"Relationship Between Test Plan, Test Cases, and Test Scripts."**~~

Test Plan vs. Test Cases:
The test plan is a high-level document that outlines the testing approach, while test cases provide detailed instructions for executing specific test scenarios.

Test Cases vs. Test Scripts:
Test cases are often written in a natural language and can be executed manually, whereas test scripts are written in a programming language and are used for automated testing.

Day 5-6:

Block 5: Bug Life Cycle

- [x] ~~**“Understanding the bug life cycle.”**~~

- New:
  - Description: The bug is identified and reported by a tester.
  - Activities: The bug is logged with details such as description, steps to reproduce, and screenshots.
- Open:
  - Description: The development team receives the bug report and begins the evaluation process.
  - Activities: Developers analyze the bug, confirm its existence, and assign it to the appropriate team member.
- In Progress:
  - Description: The assigned developer works on fixing the bug.
  - Activities: The code is modified to address the reported issue.
- Fixed:
  - Description: The developer believes the bug is fixed in the code.
  - Activities: The modified code undergoes testing by the developer to ensure the bug is resolved.
- Pending Retest:
  - Description: The fixed code is sent back to the testing team for verification.
  - Activities: Testers retest the bug to confirm whether the issue has been successfully resolved.
- Reopen:
  - Description: If the bug is still present, it is reopened and sent back to the development team.
  - Activities: Developers analyze the issue, make necessary adjustments, and repeat the bug-fixing process.
- Verified:
  - Description: The testing team confirms that the bug has been successfully fixed.
  - Activities: The bug is marked as verified, and the fix is considered effective.
- Closed:
  - Description: The bug is closed as it has been fixed, verified, and meets the quality standards.
  - Activities: The bug is marked as closed, and relevant documentation is updated.

- [x] ~~**“Importance of effective bug tracking.”**~~

- Early Detection of Issues:
  - Benefits: Bug tracking allows for the early identification of issues, reducing the chances of critical defects reaching production.

- Improved Communication:
    - Benefits: Bug tracking provides a centralized platform for communication between development and testing teams, fostering collaboration.
- Enhanced Product Quality:
    - Benefits: Identifying, tracking, and resolving bugs contribute to overall product quality and user satisfaction.
- Efficient Prioritization:
    - Benefits: Bug tracking helps prioritize issues based on severity and impact, allowing teams to address critical problems first.
- Data for Continuous Improvement:
    - Benefits: Bug tracking systems provide valuable data that can be analyzed to identify patterns, recurring issues, and areas for process improvement.
- Customer Satisfaction:
    - Benefits: Addressing and resolving bugs promptly improves customer satisfaction by delivering a more reliable and stable product.
- Traceability:
    - Benefits: Bug tracking systems provide a history of bug statuses, actions taken, and resolutions, allowing for traceability and accountability.
- Resource Optimization:
    - Benefits: Efficient bug tracking helps in optimizing resources by ensuring that efforts are focused on critical issues and preventing redundant work.

Block 6: Test Execution

- [x] ~~**"Executing test cases and reporting results."**~~

Test Execution:

- Test Environment Setup:
    - Objective: Prepare the testing environment with the necessary hardware, software, and network configurations.
    - Activities: Install the application, configure test databases, and set up any required test data.
- Test Data Preparation:
    - Objective: Ensure that the test data required for executing test cases is available and accurate.
    - Activities: Create or acquire the necessary test data to simulate real-world scenarios.
- Test Case Execution:
    - Objective: Execute the test cases according to the test plan.
    - Activities: Follow the steps outlined in each test case, inputting data, interacting with the application, and observing the results.
- Defect Logging:
    - Objective: Report any discrepancies or issues encountered during test execution.

    - Activities: If a test case fails, log a defect with detailed information, including steps to reproduce, actual and expected results, and any supporting documentation.
- Regression Testing:
    - Objective: Ensure that new changes or bug fixes do not negatively impact existing functionalities.
    - Activities: Re-run previously executed test cases, especially those related to areas affected by recent changes.
- Exploratory Testing:
    - Objective: Investigate the application beyond predefined test cases to identify any unexpected behavior.
    - Activities: Perform ad-hoc testing, exploring the application to discover issues that may not be covered by formal test cases.
- Adherence to Test Scripts (if automated):
    - Objective: Ensure that automated test scripts are executed accurately.
    - Activities: Run automated test scripts, review results, and investigate any failures.
- Documentation of Results:
    - Objective: Record the outcomes of test case execution.
    - Activities: Document whether each test case passed or failed, along with any relevant notes or observations.

Reporting Results:

- Test Execution Summary:
    - Content: High-level summary of the test execution phase.
    - Activities: Provide an overview of the number of test cases executed, passed, and failed.
- Detailed Test Results:
    - Content: Comprehensive details of individual test case results.
    - Activities: Include information such as test case ID, description, steps executed, actual results, expected results, and defect details if applicable.
- Defect Report:
    - Content: Detailed information about reported defects.
    - Activities: Provide defect status, severity, steps to reproduce, and any other relevant details. This report is crucial for the development team to address identified issues.
- Metrics and Trends:
    - Content: Analysis of test execution metrics and trends.
    - Activities: Identify patterns, recurring issues, and areas for improvement. Metrics may include pass/fail rates, defect density, and more.
- Recommendations:
    - Content: Suggestions for further testing or improvements.
    - Activities: Based on test results, offer recommendations for additional testing, areas to focus on in future releases, or process improvements.

- Test Closure Report:
  - Content: A summary report to formally conclude the testing phase.
  - Activities: Summarize overall test results, lessons learned, and recommendations. This report signifies the end of the test execution phase.

Day 7:

Block 7: Test Automation Overview

- [x] ~~**"Introduction to test automation."**~~

Key Concepts:

- Test Scripts: Automated test cases written in a scripting language.
- Test Frameworks: A set of guidelines for creating and designing test cases.
- Test Automation Tools: Software applications designed to automate the testing process.

Types of Test Automation:

- Functional Testing Automation: Validating that the software functions as expected.
- Regression Testing Automation: Verifying that new code changes don't negatively impact existing functionalities.
- Performance Testing Automation: Assessing the system's behavior under different conditions.
- Load Testing Automation: Evaluating how the system performs under specific load conditio**ns.**

- [x] ~~**"Benefits of test automation."**~~

- Efficiency and Speed:
  - Benefits: Automated tests run faster than manual tests, allowing for quicker feedback in the development process.
- Reusability:
  - Benefits: Test scripts can be reused across different phases of development and in various testing cycles.
- Consistency:
  - Benefits: Automated tests execute the same steps consistently, reducing the chance of human error.
- Increased Test Coverage:
  - Benefits: Automation allows for extensive testing of a large number of test cases, improving overall test coverage.
- Early Detection of Defects:
  - Benefits: Automated tests can be run frequently, facilitating the early detection and resolution of defects.
- Parallel Execution:

    - Benefits: Multiple test cases can be executed simultaneously, saving time and resources.
- Regression Testing:
    - Benefits: Automated tests are well-suited for frequent regression testing, ensuring that new changes do not adversely affect existing functionalities.
- Improved Accuracy:
    - Benefits: Automated tests perform tasks with precision, reducing the likelihood of human errors.

☑ ~~**“Challenges of Test Automation.”**~~

- Initial Investment:
    - Challenges: Setting up automation infrastructure and developing test scripts require an initial investment of time and resources.
- Maintenance Overhead:
    - Challenges: As the application evolves, test scripts may require frequent updates, leading to maintenance overhead.
- Limited Testing Scope:
    - Challenges: Not all testing scenarios are suitable for automation, leading to a limited testing scope.
- Script Debugging:
    - Challenges: Debugging automated test scripts can be time-consuming and complex.
- Skill Set Requirements:
    - Challenges: Automated testing may require specific skills, and not all team members may possess them.
- GUI Changes:
    - Challenges: If there are frequent changes to the application's graphical user interface (GUI), test scripts may need adjustments.
- False Positives/Negatives:
    - Challenges: Automated tests may produce false positives or negatives, requiring careful analysis to distinguish real issues from false alarms.
- Initial Learning Curve:
    - Challenges: Teams need time to learn and adapt to new automation tools and technologies.