Software Quality Engineering Course Project

Project Title: Comprehensive Quality Engineering for Open-Source Applications

Project Deadline:

The project deadline is December 07, 2025
The maximum size of a group allowed is 3.

Project Overview:

Objective: The goal of this project is to test an open-source web application/Game/integrated API application using a structured CI/CD pipeline, integrating various testing techniques, tools, and practices to ensure robust quality control. The project will involve writing automated test cases for both UI and backend code and setting up a CI/CD pipeline with continuous integration, testing, and deployment processes. A sample self-explained CI/CD pipeline is shown in the figure below.

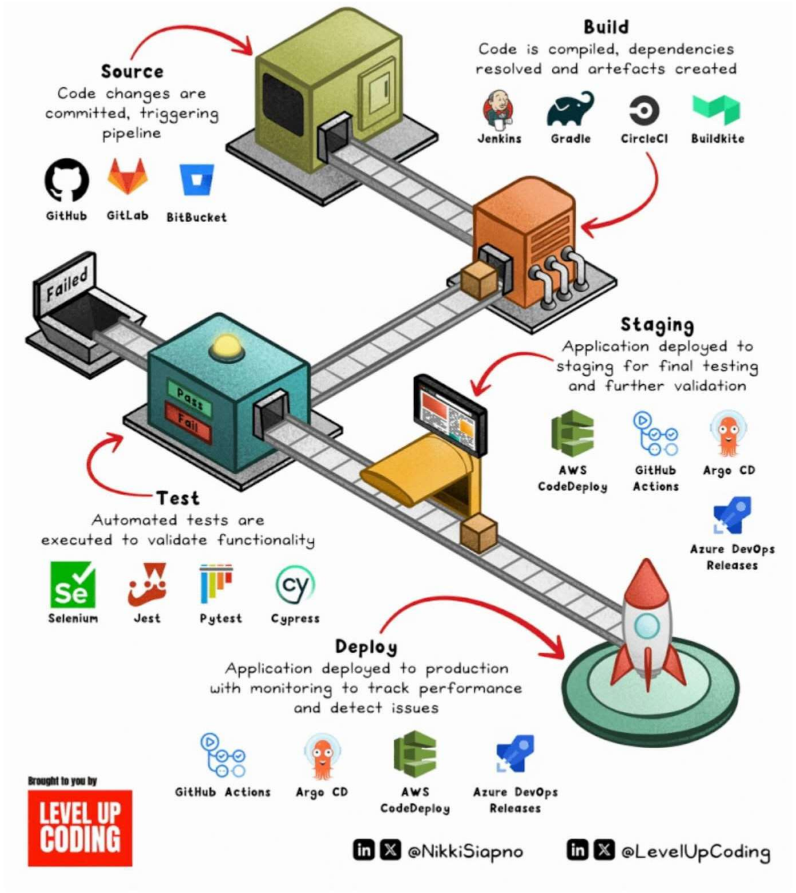
Source
Code changes are committed, triggering pipeline
GitHub
GitLab
BitBucket
Build
Code is compiled, dependencies resolved and artefacts created
Jenkins
Gradle
CircleCI
Buildkite
Failed
Pass
Fail
Staging
Application deployed to staging for final testing and further validation
AWS CodeDeploy
GitHub Actions
Argo CD
Azure DevOps Releases
Test
Automated tests are executed to validate functionality
Selenium
Jest
Pytest
Cypress
Deploy
Application deployed to production with monitoring to track performance and detect issues
GitHub Actions
Argo CD
AWS CodeDeploy
Azure DevOps Releases
Brought to you by
LEVEL UP CODING
@NikkiSiapno
@LevelUpCoding

Figure 1 : CI/CD Pipeline Explained with tools by Level up Coding.

Selection of Open-Source Application:

Choose a small to medium-sized open-source web application/Game/integrated API application that contains both backend code and a user interface (UI). The application should be actively maintained on platforms like GitHub, GitLab, or BitBucket. Examples might include projects like:

- **Jitsi Meet** (Open-source video conferencing)
- **Nextcloud** (Open-source file synchronization)
- **WordPress** (Open-source CMS)

CI/CD Pipeline Stages:

1. Source Stage (Code Repository & Triggering Pipeline)

- **Tools to Use:** GitHub, GitLab, Bitbucket, Jenkins, CircleCI
- **Description:** Set up a Git repository for the chosen open-source application and establish webhook triggers that initiate the pipeline whenever a new commit or pull request is made.
- **Implementation Steps:**
    - Clone the repository and ensure it's linked to GitHub Actions or GitLab CI to trigger builds when new changes are pushed.
    - Configure Jenkins or CircleCI to listen to changes and trigger subsequent pipeline stages.

2. Build Stage (Code Compilation & Artifact Creation)

- **Tools to Use:** Jenkins, Gradle, CircleCI, Buildkite
- **Description:** Automate the build process to compile the code, resolve dependencies, and create artifacts (e.g., Docker containers, compiled code).
- **Implementation Steps:**
    - Configure a build tool (Gradle or Maven) to compile the code and resolve dependencies.
    - Set up Jenkins to create build artifacts (JAR, WAR files, Docker images) and deploy them to staging servers.

3. Test Stage (Automated Testing)

- **Tools to Use:** Selenium, Jest, Pytest, Cypress
- **Description:** Implement automated tests for both UI and backend components of the application.
    - **UI Testing:** Use **Selenium** or **Cypress** to write tests that simulate user interactions with the application's front-end.
    - **Backend Testing:** Use **Jest** or **Pytest** to test the backend code, focusing on API endpoints and database interactions.
- **Implementation Steps:**
    - **UI Testing with Selenium/Cypress:** Write tests for key user flows like login, form submission, and navigation. Implement assertions to verify that the UI behaves as expected.

    - **Backend Testing with Pytest/Jest:** Write tests for API endpoints, database queries, and response validation. Use mock data to simulate various scenarios.
    - Integrate these tests into the pipeline to be executed automatically with every new commit or pull request.

4. Staging Stage (Final Testing & Validation)

- **Tools to Use:** AWS CodeDeploy, GitHub Actions, Argo CD
- **Description:** Deploy the application to a staging environment for final integration testing.
- **Implementation Steps:**
    - Set up **Argo CD** or **AWS CodeDeploy** to automatically deploy the application to a staging environment every time a successful build passes the testing stage.
    - Validate the staging deployment through additional manual or automated exploratory testing.

5. Deploy Stage (Production Deployment)

- **Tools to Use:** GitHub Actions, AWS CodeDeploy, Azure DevOps Releases
- **Description:** Deploy the application to production and ensure continuous monitoring and error tracking.
- **Implementation Steps:**
    - Set up **Azure DevOps Releases** or **GitHub Actions** to automate deployment to production once the staging environment has been validated.
    - Implement monitoring tools (e.g., New Relic, Sentry) to track performance and detect issues post-deployment.

Test Plan:

The test plan will cover both white-box and black-box testing, focusing on internal code structure (white-box) and functional user behaviors (black-box).

Test Plan Sections:

1. **Test Objective:**
    - Ensure that the application functions correctly in both development and production environments.
    - Perform both UI testing and backend API testing to validate the end-to-end user journey.
2. **Test Scope:**
    - **Functional Testing (black-box):** Validating core features such as login, data submission, navigation, and error handling.
    - **Non-Functional Testing:** Performance testing (load times, response times), security testing (injection attacks, XSS), and accessibility testing.
    - **Unit Testing** (white-box)**.:** Testing individual functions and methods in the backend code.
    - **Integration Testing:** Testing the interaction between different services (e.g., database, external APIs).

3. **Test Techniques:**
    - **Manual Testing:** Performed during the staging stage for exploratory testing and user experience evaluation.
    - **Automated Testing:**
        - **Unit Tests** for backend functions using Pytest or Jest.
        - **UI Tests** for user interactions using Selenium or Cypress.
4. **Test Tools and Frameworks:**
    - **Backend:** Pytest, Jest
    - **UI:** Selenium, Cypress
    - **CI/CD:** GitHub Actions, CircleCI, Jenkins, Argo CD, AWS CodeDeploy
    - **Monitoring:** New Relic, Sentry
5. **Test Environment:**
    - **Development:** Local environment with Docker containers.
    - **Staging:** Cloud-based staging server (AWS, Azure).
    - **Production:** Live production server with monitoring tools integrated.
6. **Test Cases:**
    - **UI Test Case Example:**
        1. **Test Scenario:** User logs into the application.
        2. **Steps:**
            - Navigate to the login page.
            - Enter valid credentials.
            - Click on the login button.
        3. **Expected Result:** User is successfully logged in and redirected to the dashboard.
    - **Backend Test Case Example:**
        1. **Test Scenario:** Validate the login API endpoint.
        2. **Steps:**
            - Send a POST request to the login API with valid credentials.
        3. **Expected Result:** API returns a success response with a user token.

Deliverables:

1. Test Plan Document: A comprehensive test plan detailing all manual and automated test cases for both white-box and black-box testing as per IEEE Standard.

2. CI/CD Pipeline Configuration: Configuration files for Jenkins, CircleCI, or GitHub Actions to automate the pipeline and tests.

3. Test Results & Reports: Include results from unit tests (white-box) and UI/API tests (black-box), along with any issues discovered during testing.

4. Deployment Instructions: Documentation on how to deploy the application to the staging and production environments using CI/CD tools.

Evaluation Criteria:

1. Quality of Test Plan: Comprehensive coverage of both white-box and black-box testing techniques.

2. Test Coverage: The percentage of code covered by unit tests (white-box) and user scenarios tested (black-box).

3. Tool Integration: Effective use of testing tools in the CI/CD pipeline.

4. Test Execution: Successful execution of tests in automated pipelines, with issues being tracked and resolved.

5. Deployment Success: Correct deployment to staging and production environments with minimal issues.

Marking Rubrics:

Test Plan Quality (20%): Comprehensiveness of the test plan, including white-box and black-box tests. Clear definition of testing techniques and detailed test cases.

Test Coverage (20%): Percentage of application functionality covered by both automated unit tests (white-box) and user tests (black-box).

Tool Integration (15%): Effective use of CI/CD tools (e.g., Jenkins, CircleCI, GitHub Actions) for automated testing and deployment processes.

Test Execution (15%): Successful execution of tests in automated pipelines. Addressing and tracking issues that arise during testing.

Documentation and Deliverables (10%): Quality of the documentation provided, including the test cases document, CI/CD configuration, test reports, and deployment instructions.

Deployment and Monitoring (10%): Successful deployment of the application to staging and production environments, with correct monitoring and error tracking in place.

Team Collaboration and Progress (10%): Collaboration, communication, and progress tracking within the team. Meeting deadlines and delivering work as per the plan.