NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCE

Software Quality Engineering

TestOps: Framework Evaluation

GROUP MEMBERS

Name	Roll No
Emaan Fatima	22F- 3640
Nehal	22F- 3638

Executive Summary

This report examines the concept of TestOps, focusing on its importance in streamlining modern testing workflows. It identifies the gaps in traditional systems. By integrating TestOps, organizations can achieve greater scalability, efficiency and collaboration in their testing processes.

Introduction

In today's fast-paced software development environment, the demand for faster delivery cycles, increased automation, and consistent quality has never been higher. Organizations are transitioning from traditional testing methods to approaches that seamlessly integrate testing into the software development lifecycle. This transition has given rise to **TestOps**, a practice that combines testing, operations, and automation to ensure quality assurance remains efficient, scalable, and aligned with modern development practices.

TestOps

TestOps, test operations, is an approach that integrates testing practices with business processes to facilitate quality assurance (QA) in software development It distinguishes between development (Dev), quality (QA), and operations (Ops) edge of the network, ensuring SDLC testing) is a consistent, efficient, and collaborative part.

TestOps Needs

1. **Streamlining and automating testing processes** to keep up with continuous integration and deployment (CI/CD) pipelines.
2. **Collaborating effectively** between testers, developers, and operations teams to ensure seamless communication and shared responsibility.
3. **Monitoring and analyzing test results continuously** for better decision-making and rapid feedback.
4. **Reduced execution time** TestOps allows developers to leverage the benefits of test automation, it eliminates the complexity of manual testing, and enables developers to make more effective use of their test environments.

Components of TestOps

1. **Planning:** TestOps planning identifies and prioritizes what will be tested, how (including the test environment), when, and by whom.
2. **Management**: TestOps management helps ensure the testing processes are efficient and scalable via tools that enhance visibility and collaboration.
3. **Execution:** TestOps execution is the actual process of software testing.
4. **Analysis:** TestOps analysis involves reviewing various aspects of the testing operation, including performance, stability, failure diagnostics, and more, to inform improvements in the testing process.

Adopting TestOps in One's Organization

⇨ **Establish the Right Tools and Infrastructure**

- Implement CI/CD tools like GitLab CI, or GitHub Actions.
- Use automated testing frameworks (e.g., Selenium, Playwright) for functional and UI tests.
- Use performance testing tools (e.g., K6, JMeter, Locust) and integrate them into pipelines.

⇨ **Promote Collaboration**

- Foster communication between QA, developers, and operations teams.
- Use tools like Slack or Microsoft Teams for instant updates and feedback.

⇨ **Shift-Left Testing Approach**

- Involve testing early in the development process.
- Use techniques like unit testing, API testing, and code analysis in early stages.

Mapping TestOps Needs to our Frameworks

- **UI Test Automation**

  Achieved:

  ⇨ **Automated Test Execution:**

  Integrated with CI/CD pipelines for seamless test execution across multiple environments.

  ⇨ **Cross-Browser Compatibility:**

  Automated tests verified functionality on Chrome, Firefox, and Edge, ensuring consistent user experiences.

  ⇨ **Reporting:**

  Provided pass/fail results in the console output, enabling quick feedback on test outcomes.

  Gaps:

  ⇨ **Centralized Dashboard:**

  No unified interface to track execution across multiple browsers and environments.

- **Unit Test Automation**

  Achieved:

  ⇨ **Test Parameterization & Data-Driven Testing**:

  Automated tests are data-driven, taking test data from Excel file to ensure flexibility in testing different inputs and scenarios.

  ⇨ **Reporting:**

  Provided pass/fail results in the console output, enabling quick feedback on test outcomes.

  Gaps:

  ⇨ **Continuous Feedback and Monitoring:**

  Lack of integration with monitoring tools for real-time tracking of test execution in CI/CD pipelines, such as feedback on performance

- **Web API and Performance Test Automation**

Achieved:

⇨ **Automated Test Execution**:

- Tests for the four RESTful API endpoints (GET, POST, PUT, DELETE) are fully automated using **Jest** and **Frisby**.
- Tests are integrated with CI/CD pipelines, ensuring automated execution during each build.
- Load testing scenarios are automated using **K6**, simulating virtual users and testing performance under load.

⇨ **Reporting:**

Provided pass/fail results in the console output, enabling quick feedback on test outcomes.

⇨ **Error Handling:**

Both valid and invalid test scenarios are covered, ensuring that error responses are properly tested.

⇨ **Performance Benchmarking**:

- The performance tests provide benchmarking data, such as response times, throughput, and system performance under load.