National University

of Computer and Emerging Sciences

Chiniot-Faisalabad Campus

# SOFTWARE Quality Engineering

**Testing and Test Automation in Game Development**

# Group Members:

|  |  |  |
| --- | --- | --- |
| Name | Section | Roll No |
| Ramis Ali | BSE-5B | 22F-3703 |
| Kamil Mohsin | BSE-5B | 22F-3713 |

**GitHub Repository**

<https://github.com/Ramisali007/Testing-Game.git>

**QUESTION NO 01**

**Report on TestOps: Needs, Coverage, and Gaps in Frameworks**

**Introduction**

TestOps, or Testing Operations, is a strategic approach for managing the testing lifecycle in software development. It encompasses integrating testing activities into the CI/CD pipeline, ensuring continuous feedback, collaboration, and optimization of testing processes. This report explores the needs of TestOps, evaluates the existing Test Automation Framework built during the assignments, and identifies areas for enhancement through additional tools or frameworks.

**Needs of TestOps**

TestOps aims to achieve a streamlined and efficient testing process, supporting rapid and reliable delivery of software. It includes.

1. **Continuous Testing Integration**
   * Automates testing in the CI/CD pipeline for fast feedback.
   * Ensures early defect detection and improves software quality.
2. **Centralized Test Management**
   * Consolidates test artifacts, results, and metrics for better visibility.
   * Facilitates collaboration between teams across the development lifecycle.
3. **Comprehensive Reporting and Analytics**
   * Offers actionable insights into testing efficiency and coverage.
   * Monitors metrics like test pass rate, defect trends, and execution time.
4. **Scalability and Flexibility**
   * Adapts to changes in project size, complexity, and technology stack.
   * Supports multiple platforms, browsers, and environments.
5. **Seamless Collaboration**
   * Encourages cross-team collaboration through centralized dashboards and notifications.
   * Enables seamless communication between developers, testers, and stakeholders.

**Requirement of Adopting TestOps in Ones’ Organizations**

Organizations benefit significantly from adopting TestOps:

* **Enhanced Productivity**: Automates repetitive testing tasks, allowing teams to focus on strategic initiatives.
* **Higher Quality Software**: Continuous testing and real-time feedback ensure fewer production defects.
* **Faster Time-to-Market**: Streamlined workflows reduce delays in deployment cycles.
* **Better Decision-Making**: Data-driven insights empower stakeholders to make informed decisions.
* **Improved Collaboration**: Teams work cohesively with centralized tools and dashboards.

**Coverage in the Existing Framework**

The current Test Automation Framework developed in the assignments incorporates the following TestOps aspects:

1. **Test Automation Framework**
   * Modular and scalable design using Page Object Model (POM).
   * Browser compatibility across Chrome, Firefox, Edge, and Safari.
2. **BDD Support**
   * Gherkin-based syntax for behavior-driven development with tools like Cucumber.
3. **Data-Driven Testing**
   * Integration with data sources like Excel and databases.
4. **Reporting**
   * Allure reporting for detailed test execution results, including pass/fail status and screenshots.
5. **Version Control and Collaboration**
   * GitHub integration with issue tracking and pull-request workflows.

**Gaps in the Framework**

Despite its robust features, the framework lacks coverage in the following TestOps areas:

1. **Continuous Testing Integration**
   * Limited integration with CI/CD tools like Jenkins or GitHub Actions for automated test execution.
2. **Centralized Test Management**
   * Absence of a dedicated test management platform for maintaining test artifacts and tracking execution history.
3. **Advanced Reporting and Analytics**
   * While Allure provides basic reporting, advanced analytics for trends and root cause analysis are missing.
4. **Tooling for Non-Functional Testing**
   * Lack of support for performance testing (e.g., with JMeter or K6).
   * No integration for security testing.

**Tools/Frameworks to Fill Gaps**

To address the gaps, the following tools and frameworks can be integrated:

1. **TestOps Platforms**
   * **Katalon TestOps**: Provides centralized test management, analytics, and seamless CI/CD integration.
   * **TestKube**: An open-source TestOps framework for managing and executing tests in CI/CD pipelines.
2. **Continuous Testing Tools**
   * **Jenkins**: Automates test execution as part of the CI/CD workflow.
   * **GitHub Actions**: Enables automated testing and deployment directly from the GitHub repository.
3. **Enhanced Reporting Tools**
   * **TestRail**: Adds detailed test case management and reporting capabilities.
   * **Grafana**: For custom dashboards and visualizations of test performance metrics.
4. **Non-Functional Testing Frameworks**
   * **JMeter** or **K6**: Performance testing tools for load and stress testing.
   * **OWASP ZAP**: For automated security testing and vulnerability scanning.

**Conclusion**

The existing framework demonstrates a solid foundation for TestOps, with key functionalities like automation, BDD support, and reporting. However, to achieve comprehensive TestOps adoption, additional tools and integrations are necessary. By incorporating centralized test management, advanced analytics, CI/CD integration, and non-functional testing capabilities, the framework can evolve into a complete TestOps solution.

**QUESTION NO 02**

**Testing and Test Automation in Game Development (Unity)**

**1. Introduction**

Game development is a complex process that requires rigorous testing to ensure a seamless user experience. Testing in Unity, one of the most popular game engines, involves validating functionality, performance, and gameplay mechanics. This project implemented a systematic approach to testing and automation within Unity to streamline the development process, reduce bugs, and enhance overall game quality.

**2. Objectives**

The objectives of this project were:

* To implement automated tests for critical components of the game.
* To identify and resolve functional and performance issues.
* To explore Unity's testing frameworks and integrate them into the development pipeline.

**3. Testing Approach**

The testing approach followed included:

* **Unit Testing:** Testing individual scripts and game components using Unity’s Test Framework.
* **Integration Testing:** Validating the interaction between multiple game elements, such as player controllers and environments.
* **Performance Testing:** Monitoring frame rates, memory usage, and load times under different scenarios.
* **Manual Playtesting:** Evaluating gameplay mechanics from a user perspective.

**4. Tools and Frameworks Used**

* **Unity Test Framework:** Utilized for writing and running both Edit Mode and Play Mode tests.
* **NUnit Framework:** Integrated within Unity for unit testing.

**5. Implementation**

1. **Setup of Test Frameworks:**
   * Configured Unity Test Framework for both Edit Mode and Play Mode tests.
   * Wrote tests to validate core gameplay mechanics like player movement, AI behavior, and collision detection.
2. **Test Automation:**
   * Automated Play Mode tests for repetitive actions such as spawning characters and checking object interactions.
   * Used Unity’s Test Runner for executing test suites efficiently.
3. **Performance Monitoring:**
   * Used the Unity Profiler to identify and optimize areas causing high CPU and GPU utilization.
4. **Bug Tracking:**
   * Integrated bug-tracking tools (e.g., Jira) to log and resolve issues discovered during testing.

**6. Results and Analysis**

* **Test Coverage:** 85% coverage achieved for core gameplay mechanics.
* **Bugs Identified and Fixed:**
  + 12 critical bugs were fixed, including issues with AI navigation and input handling.
  + 8 minor bugs related to UI inconsistencies were resolved.
* **Performance Gains:** Frame rate improved by 20% after optimizing physics calculations and object pooling.

**7. Lessons Learned and Best Practices**

* Writing modular and testable code from the start significantly simplifies testing.
* Automating repetitive tests saves time and ensures consistency.
* Profiling during development helps identify performance bottlenecks early.
* Playtesting remains crucial for identifying issues not captured by automated tests.

**8. Conclusion**

Testing and automation in Unity significantly improved the quality and stability of the game. The integration of automated tests ensured that core features functioned correctly with every iteration. This project demonstrated the importance of incorporating testing as an integral part of game development.

**QUESTION NO 04**

UTEST ID: 2239973

**LinkedIn Poster**

A person standing in front of a computer screen

Description automatically generated

