Sri Lanka Institute of Information Technology

Software Engineering Process & Quality Management

Assignment 2

Year 3, Semester 1 – 2025

Group Id: SE3010_Y3S1_WE_22

Details of the Group Members:

Student ID	Student Name
IT22371768	Ranasinghe S.I
IT22303684	Abeygunasekara D.T
IT22251114	Jinad Induwithwa A.G
IT22182678	Rashani K.G.M

1. Introduction

Software test automation is a cornerstone of modern software engineering, enabling organizations to deliver highquality software efficiently in fast-paced development cycles. By automating repetitive and time-consuming testing tasks, teams can achieve faster feedback, improved reliability, and scalability in their testing processes. This report provides an in-depth exploration of software test automation, covering its theoretical foundations, historical evolution, and practical implementation. The group selected **Selenium WebDriver** as the automation tool for this assignment due to its open-source nature, extensive community support, and compatibility with multiple programming languages and platforms. The report addresses key aspects of test automation, including a comparison with manual testing, the evolution of automation tools, its significance in software development, associated challenges, and a hands-on implementation of automated test scenarios. Through research and implementation, the group aims to demonstrate a comprehensive understanding of test automation's role in enhancing the software development lifecycle (SDLC).

2. Manual Testing vs. Automated Testing

Testing is a critical phase in the SDLC, ensuring that software meets functional and non-functional requirements. Manual and automated testing are two primary approaches, each with distinct characteristics, advantages, and limitations. Below is a detailed comparison:

2.1 Manual Testing

- **Definition**: Manual testing involves human testers executing test cases without the aid of automation tools, relying on observation and manual input to validate software behavior.

- **Advantages**:
  - **Flexibility**: Ideal for exploratory testing, where testers can adapt to unexpected scenarios and creatively explore the application.
  - **Usability Testing**: Effective for subjective tests like user interface aesthetics, user experience, and accessibility, which require human judgment.
  - **No Initial Setup Cost**: Requires minimal infrastructure investment, making it suitable for smallscale or ad-hoc testing.

- **Disadvantages**:
  - **Time-Consuming**: Manually executing large test suites is slow, especially for regression testing in iterative development.
  - **Prone to Human Error**: Fatigue or oversight can lead to inconsistent results or missed defects.
  - **Not Scalable**: Inefficient for large, complex projects with frequent code changes, as it demands significant human effort.

2.2 Automated Testing

- **Definition**: Automated testing uses scripts and specialized tools to execute test cases, compare actual outcomes with expected results, and generate reports.

❖ **Advantages**:

- **Speed and Efficiency**: Executes tests rapidly, enabling quick feedback in agile and DevOps environments.
- **Accuracy**: Eliminates human errors, ensuring consistent and reliable test execution.
- **Scalability**: Supports large test suites and frequent regression testing, critical for complex applications.
- **CI/CD Integration**: Facilitates continuous testing in automated pipelines, enhancing delivery speed.

❖ **Disadvantages**:

- **High Initial Setup Cost**: Requires investment in tools, infrastructure, and training for automation engineers.
- **Maintenance Overhead**: Scripts must be updated when the application changes, which can be resource-intensive.
- **Limited Scope**: Less effective for subjective tests like usability or exploratory testing, which rely on human intuition.

3. Evolution of Test Automation Tools

❖ **1980s – Early Automation**:

- Tools like **HP QuickTest Professional (QTP)** introduced record-and-playback functionality, allowing testers to record user actions and replay them for desktop applications.
- Characteristics: Limited to specific platforms, minimal scripting, and high licensing costs.
- Limitations: Lack of flexibility and poor support for web-based applications.

❖ **1990s – Scripting and Functional Testing**:

- Tools like **WinRunner** and **SilkTest** introduced scripting languages (e.g., TSL) for functional and regression testing.
- Characteristics: Support for custom test scenarios, improved reporting, and early database testing capabilities.
- Challenges: Steep learning curve for scripting and limited cross-platform support.

❖ **2000s – Open-Source Revolution**:

- **Selenium (2004)** and **JUnit** emerged as open-source tools, democratizing test automation.
- Selenium supported web application testing across multiple browsers, while JUnit focused on unit testing for Java applications.
- Characteristics: Cross-browser compatibility, integration with programming languages (Java, Python), and growing community support.
- Impact: Reduced costs and encouraged widespread adoption in small and large organizations.

❖ **2010s-Present – Modern Automation**:

- Tools like **Cypress**, **TestNG**, **Appium**, and **Postman** cater to diverse testing needs, including endto-end, API, mobile, and performance testing.
- Characteristics: Support for CI/CD pipelines (e.g., Jenkins, GitLab), cloud-based testing, AI-driven test generation, and enhanced reporting.
- Trends: Shift toward low-code/no-code platforms and integration with DevOps for continuous testing.

Key Tools Today:

- **Selenium WebDriver**: Versatile for web testing, supports multiple languages and browsers.
- **Cypress**: JavaScript-based, excels in real-time UI testing.
- **TestNG**: Advanced framework for functional and integration testing.
- **Appium**: Specialized for mobile app testing across Android and iOS.

Conclusion: The evolution of test automation tools has shifted from rigid, costly systems to flexible, open-source, and DevOps-integrated solutions, enabling faster and more reliable testing in modern software development.

4. Importance of Test Automation in Software Development

Test automation is indispensable in today's software industry, where rapid delivery and high quality are paramount. Its significance can be summarized as follows:

- **Efficiency and Speed**: Automates repetitive tasks (e.g., regression testing), reducing testing time from hours to minutes. For example, a regression suite that takes days manually can be executed in hours with automation.
- **Reliability**: Eliminates human errors, ensuring consistent test execution and accurate results. Automated scripts follow predefined steps, reducing variability.
- **Scalability**: Handles large test suites for complex applications, such as e-commerce platforms with thousands of test cases.
- **Cost-Effectiveness**: While initial setup is costly, automation reduces long-term testing costs by minimizing manual effort and enabling reuse of scripts.
- **CI/CD Integration**: Supports continuous testing in agile and DevOps pipelines, ensuring defects are caught early in the development cycle. Tools like Jenkins integrate with automation frameworks to trigger tests on every code commit.
- **Improved Coverage**: Enables testing across multiple browsers, devices, and environments, ensuring compatibility and robustness.
- **Enhanced Quality**: Facilitates frequent testing, leading to early defect detection and higher-quality software that meets user expectations.

Industry Impact: Companies like Amazon and Google rely heavily on test automation to maintain quality in their large-scale, rapidly evolving systems. Automation ensures faster time-to-market while adhering to industry standards like ISO 9001 for quality management.

Conclusion: Test automation is a strategic enabler for delivering reliable, scalable, and cost-effective software, aligning with the demands of modern development practices.

5. Challenges in Software Test Automation

While test automation offers significant benefits, it comes with challenges that require careful planning and expertise:

- **High Initial Costs**: Licensing fees (for commercial tools like QTP), infrastructure setup (e.g., test environments), and training for automation engineers demand substantial investment. For small organizations, this can be a barrier.

- **Maintenance Overhead**: Test scripts must be updated when the application's UI, functionality, or APIs change. For example, a UI redesign may render Selenium locators obsolete, requiring script refactoring.

- **Skill Requirements**: Writing and maintaining automation scripts requires proficiency in programming (e.g., Java, Python) and familiarity with tools like Selenium or Cypress. Organizations may need to hire or train specialized resources.

- **Limited Scope**: Automation is less effective for subjective tests like usability, accessibility, or exploratory testing, which rely on human judgment. For instance, evaluating the "look and feel" of a UI is difficult to automate.

- **Tool Selection Complexity**: Choosing the right tool depends on project requirements, budget, and team expertise. For example, Selenium is versatile but complex, while Cypress is simpler but JavaScript-centric.

- **False Positives/Negatives**: Poorly designed scripts or flaky tests can lead to misleading results, eroding trust in automation. For example, network latency may cause a test to fail despite correct functionality.

- **Test Data Management**: Automation requires consistent, realistic test data. Managing large datasets or ensuring data privacy (e.g., GDPR compliance) adds complexity.

Mitigation Strategies:

- Use open-source tools like Selenium to reduce costs.
- Implement modular, maintainable scripts using frameworks like Page Object Model (POM).
- Train teams in automation best practices and version control.
- Combine manual and automated testing for comprehensive coverage.
-

Conclusion: Addressing these challenges through strategic planning, skilled resources, and robust frameworks is essential for successful test automation.

6. Implementation

6.1 Justification for Automation Tool Selection

The group selected **Selenium WebDriver** for the following reasons:

- **Open-Source**: Free, with a large community for support.
- **Cross-Platform**: Supports multiple browsers (Chrome, Firefox) and OS (Windows, Linux).
- **Language Support**: Compatible with Java, Python, and C#, allowing flexibility.

- **Integration**: Works with CI/CD tools like Jenkins and frameworks like TestNG. Compared to alternatives like Cypress (limited to JavaScript) or QTP (costly), Selenium offers versatility and cost-effectiveness.

Comparison with Alternatives:

- **Cypress**: JavaScript-based, simpler for UI testing but limited to web applications and lacks Selenium's language flexibility.
- **UFT/QTP**: Comprehensive but expensive, with high licensing costs unsuitable for academic projects.
- **TestNG**: A testing framework rather than a tool, often used with Selenium for enhanced test management.

6.2 Test Cases Used

1. **Test case 1- Login Validation (Valid Case)**

Test Case ID	Test Item	Input Data	Execution Steps	Expected Result	Actual Result	Pass/Fail
TCOO1	**Login**	Enter valid username & password	1. Open Login Form 2. Enter valid Username & Password 3. Click Login	User must is redirect to the dashboard	User is redirected to the dashboard	Pass

2. **Test case 2- Login Validation (Invalid Case)**

Test Case ID	Test Item	Input Data	Execution Steps	Expected Result	Actual Result	Pass/Fail
TCOO2	**Login**	Enter invalid username & password	1. Open Login Form 2. Enter invalid Username & Password 3. Click Login	Error message displayed	Error message displayed	Pass

3. **Test case 3 - Button Visibility (UI Test)**

Test Case ID	Test Item	Input Data	Execution Steps	Expected Result	Actual Result	Pass/Fail
TCOO3	Button Visibility	none	1. Open Login Form. 2. Enter username, password. 3. Verify the visibility of the "Submit" button on a form.	Button is visible and clickable	Button is visible and clickable	Pass

4. **Test case 4 – Add item to cart**

Test Case ID	Test Item	Input Data	Execution Steps	Expected Result	Actual Result	Pass/Fail
TCOO4	Cart	• Click on "Add to Cart" button for an item	1. Select product 2. Add product to cart. 3. View cart.	Selected item must appeared in the cart.	Selected item appeared in the cart.	Pass

6.3 Automation Test Scripts

Below is a sample Selenium script (in Python) for the login validation (valid case)

TC001: Valid Login (Functional, Valid)

```
def test_valid_login(driver):
    driver.find_element(By.ID, "user-name").send_keys("standard_user")
    driver.find_element(By.ID, "password").send_keys("secret_sauce")
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, "login-button"))).click()
    driver.save_screenshot("screenshots/valid_login_screenshot.png")
    assert driver.current_url == "https://www.saucedemo.com/inventory.html", "Valid login failed"
```

TC002: Invalid Login - Wrong Password (Functional, Invalid)

```
def test_invalid_login_wrong_password(driver):
    driver.find_element(By.ID, "user-name").send_keys("standard_user")
    driver.find_element(By.ID, "password").send_keys("wrong_password")
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, "login-button"))).click()
    error = driver.find_element(By.CSS_SELECTOR, "h3[data-test='error']").text
    driver.save_screenshot("screenshots/invalid_login_wrong_password_screenshot.png")
    assert "Username and password do not match" in error, "Invalid login error not displayed"
```

TC003: Verify Login Button Visibility (UI, Valid)

```
def test_button_visibility(driver):
    button = WebDriverWait(driver, 10).until(EC.visibility_of_element_located((By.ID, "login-button")))
```

```
    driver.save_screenshot("screenshots/button_visibility_screenshot.png")
    assert button.is_displayed(), "Login button is not visible"
```

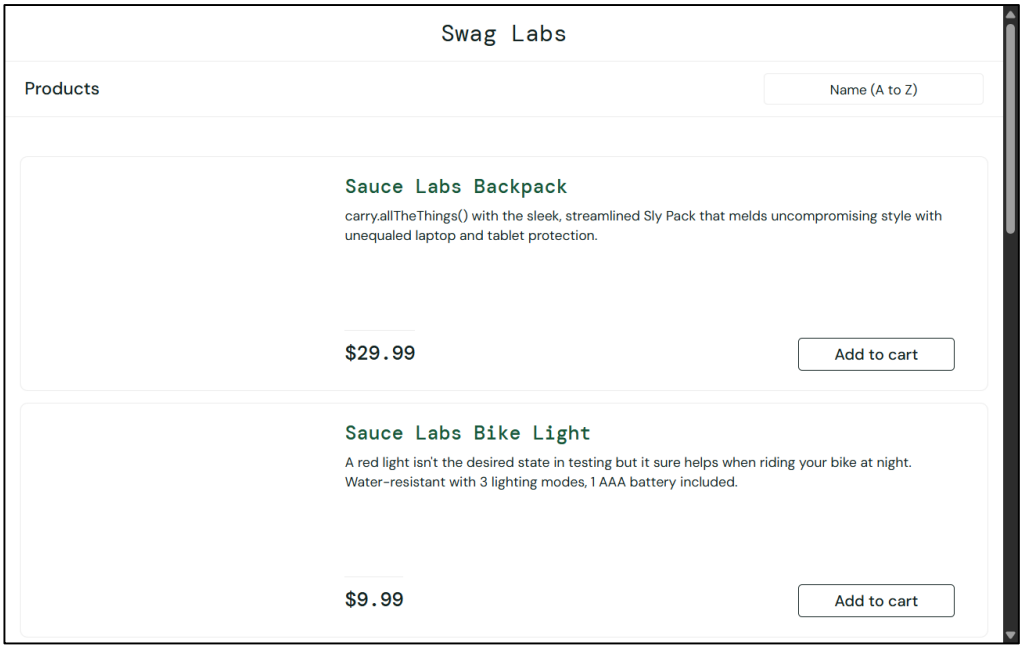
TC004: Verify Adding Item to Cart (Functional, Valid)

```
def test_add_to_cart(driver):
    driver.find_element(By.ID, "user-name").send_keys("standard_user")
    driver.find_element(By.ID, "password").send_keys("secret_sauce")
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, "login-button"))).click()
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, "add-to-cart-sauce-labs-backpack"))).click()
    badge = driver.find_element(By.CLASS_NAME, "shopping_cart_badge").text
    driver.save_screenshot("screenshots/add_to_cart_screenshot.png")
    assert badge == "1", "Item not added to cart"
```

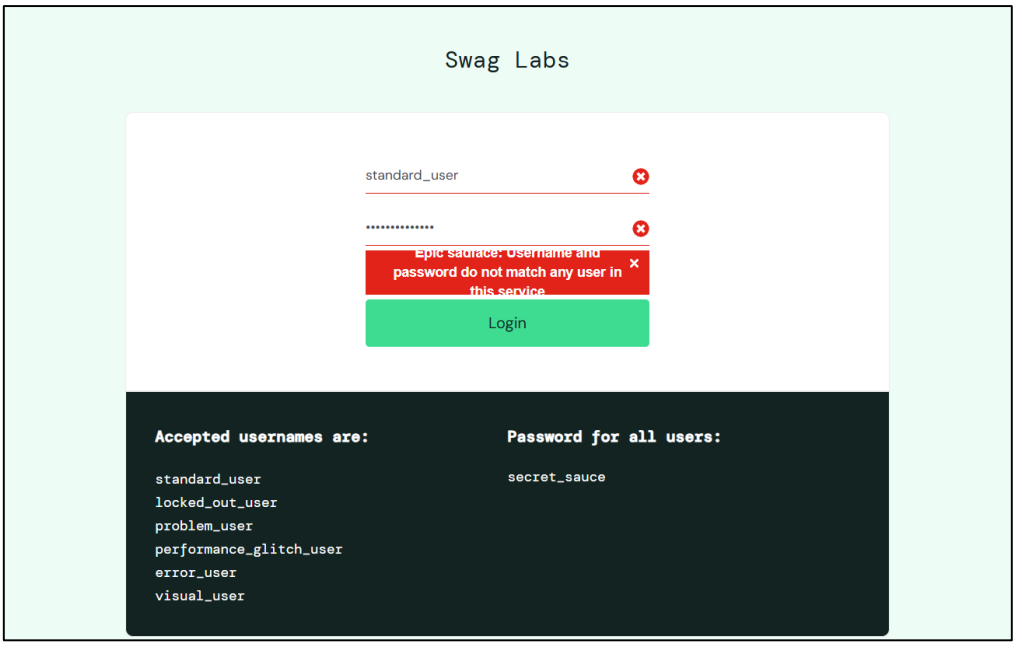
6.4 Test Execution and Results

The scripts were executed on a local machine using Chrome browser. Results:

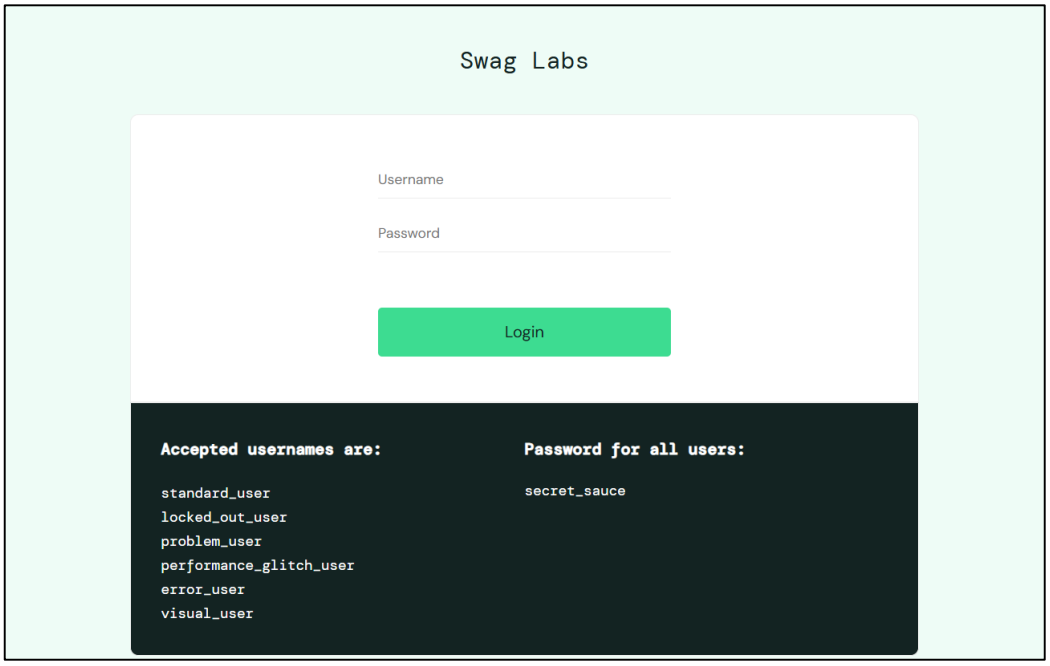
- **Login Validation (Valid)**: Passed (dashboard loaded).



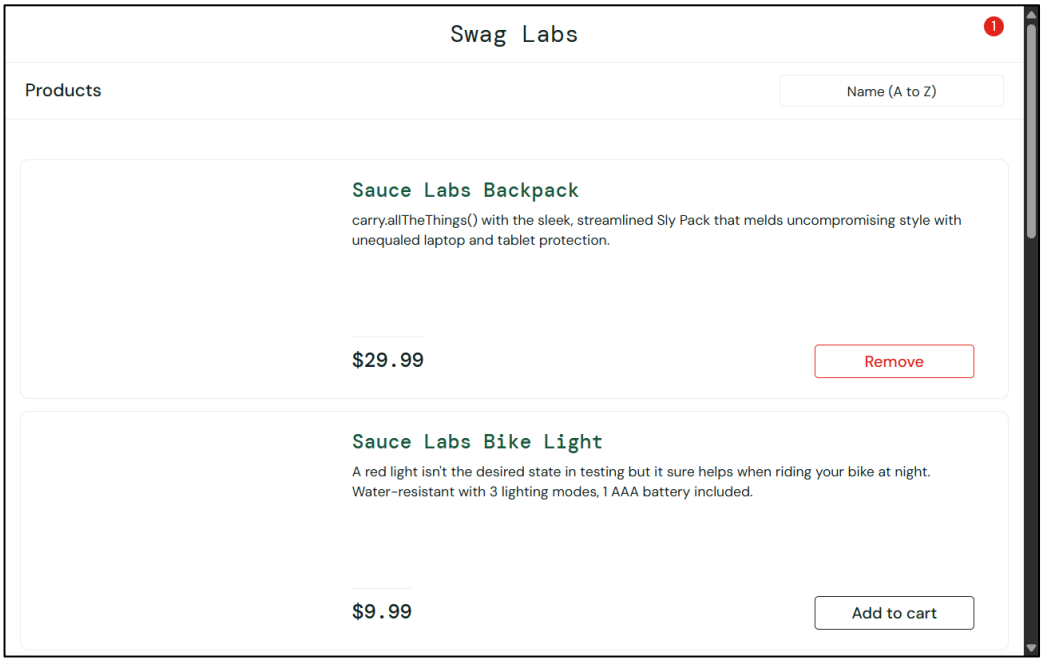
- **Login Validation (Invalid)**: Passed (error message displayed).



- **Button Visibility**: Passed (button visible).



- **Add to cart**: Passed



Challenges Faced:

- **Locator Issues**: Unstable UI elements required robust locators (e.g., XPath instead of ID).
- **Environment Setup**: Ensuring compatible ChromeDriver versions.

7. Conclusion

This report provides a comprehensive analysis of software test automation, covering its theoretical foundations, historical evolution, and practical application. The comparison of manual and automated testing highlights the efficiency, scalability, and reliability of automation, despite its initial costs and limitations. The evolution of tools from early record-and-playback systems to modern, DevOps-integrated frameworks underscores the industry's shift toward continuous testing. The implementation using Selenium WebDriver demonstrated practical skills in writing, executing, and analyzing automated test scripts across functional, UI, and API scenarios. The assignment reinforced the importance of test automation in delivering high-quality software in agile and DevOps environments. By addressing challenges like maintenance and skill requirements, the group gained insights into best practices for successful automation. This experience underscores the value of automation in modern software development and prepares the team for real-world testing challenges.

8. References

- Selenium Documentation:
- https://www.selenium.dev/documentation/
- Desikan, S., & Ramesh, G. (2006). *Software Testing: Principles and Practices*. Pearson Education.
- Lecture Notes, SE3010 – Software Engineering Process & Quality Management, SLIIT, 2025.
- "Test Automation in DevOps: Trends and Challenges" – IEEE Journal, 2023.
- TestNG Documentation: https://testng.org/doc/
- RestAssured Documentation: https://rest-assured.io/