IMPROVING RELIABILITY OF THE CROSSFIRE ELECTRONIC DATA INTERCHANGE (EDI) SYSTEM BY DEPLOYING PLAYWRIGHT FOR FUNCTIONAL TESTING

Michelle Mejos Belagtas and Shahid Ali

Department of Information Technology AGI Institute,
Auckland, New Zealand

ABSTRACT

This research study aimed at improving the testing process for the Crossfire EDI System. The traditional manual functional testing experienced to be slow, repetitive, and at the same time can be prone to human error. Following the structured Software Testing Lifecycle (STLC) within the agile Scrum framework, this study sees Playwright as the primary automation tool due to its modern capabilities for web app testing. The results were significant that it shows automation dramatically reduced the manual effort without compromising the accuracy.

KEYWORDS

Electronic data interchange, Agile, Software Testing Lifecycle, Playwright, Test Automation Framework.

1. INTRODUCTION

Sandfield is a New Zealand-based software company that provides reliable solutions for its clients. And one of its offered solutions is the Crossfire Electronic Data Interchange (EDI) system. The EDI system helps business clients to exchange order-related documents, such as invoices, purchase orders, and shipment details, in a digital format instead of paper. It works by translating documents into the required file formats such as XML, CSV, JSON, and EDIFACT, so that suppliers, retailers, logistics providers, and other partners can easily share information.

In simple term, the EDI system acts like a middleman between businesses, making sure that transactions are sent and received correctly. All documents are automatically recorded and categorized, which prevents them from being lost or duplicated. Research shows that EDI makes business-to-business transactions much faster, what used to take hours or even days can now be done in seconds [1]. This also reduces accidental human errors, such as incorrect invoice amounts or missed orders. And also provides real-time tracking of orders, shipments, and product status, giving businesses clear visibility into their supply chain.

The Crossfire EDI system is catering to hundreds of clients with different business requirements and demands. This makes EDI powerful because it can customize the existing system based on

David C. Wyld et al. (Eds): MLNLP, ASOFT, CSITY, NWCOM, SIGPRO, AIFZ, ITCCMA – 2025
pp. 207-244, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.152017

the client's request. Simply, the Crossfire development team works by integrating code changes or new features into the system regularly.

However, with the growing clients of the Crossfire EDI system, the manual approach to functional testing of pre- and post-deployment makes it time-consuming for the developers. Furthermore, a manual practice is prone to error, especially when dealing with a huge validation checklist. A frequent customer service call and emails were evident after a deployment change, because of incomplete and unreliable manual testing. Making the developers roll back changes and produce more hours of work to fix the error. These errors could have been detected early with automation testing. Automation is needed to ensure that changes to the system do not break important functions and that issues are detected early.

Since the EDI system is continuously maintained and upgraded to support new business requirements, integrate additional trading partners, and comply with changing industry standards, every system release or update carries potential risks. Major partners, such as Woolworths, Mitre10, and Mainfreight, rely heavily on Sandfield's EDI system, so any failure, bug, or defect can directly affect services, potentially delaying orders and invoices and disrupting the logistics chain. Specific risks include failed message processing, misconfigured connection parameters, system downtime, and undetected errors due to manual validation checks.

This EDI system issue isn't just a technical hiccup; it impacts real people and real businesses. Specifically, the developers and integration teams are being slowed down by the repetitive, time consuming tasks, which reduces their ability to innovate. Also, business clients depend on the EDI system for functional operations where failures can disrupt or delay business processes. Lastly, the organisation risks work inefficiency and higher operational costs if validation remains manual. Manual checks are a drag on speed and reliability. Automating the process makes testing reliable because scripts can complete the checks in a more consistent way. It also helps find problems before they affect the system and allows teams to spend more time on important tasks instead of repeating the same checks.

Moreover, this research study focuses on automating pre- and post-deployment functional validation testing for the Crossfire EDI system. The scope includes the five checklists mentioned below:

- Test Design and Execution for the Crossfire EDI System Using Playwright: To design and execute test scripts using Playwright automation tool. That is reusable and maintainable for the EDI system, which reduces human error and saves testing time.
- Pre-Deployment Functional Testing for the Crossfire EDI System: To automate the predeployment task that captures a list of log errors, errored messages, and errored interchanges, to make sure that the system is stable before the actual system change deployment.
- Post-Deployment Functional Testing for the Crossfire EDI System: To automate functional testing tasks after deployment to confirm that any changes introduced to the existing system will detect defects as soon as possible.
- Testing of the Crossfire EDI System: To make sure that all critical workflow functionality, including the admin configuration is tested which can impact the major system functions. This includes the message processing, core admin functions.
- Generating Automated Test Results on the Crossfire EDI System: To capture testing report, and capture system log errors, which helps developers to identify and address issues quickly.

This research is organized as follows. Section 2 represents literature review. Section 3 presents the methodology for this research. Section 4 covers the execution of this research study. Section 5 represents the results. Section 6 represents the discussion of the research; section 7 covers the conclusion of the research study and in Section 8 future recommendations are covered.

2. LITERATURE REVIEW

Software testing is necessary for every existing system nowadays, because people rely on it to do the work. Many studies have looked at different automation tools, testing methods, and techniques to make the testing better. This review shows related research findings and points out gaps at the end of the literature review that can help guide us in deploying Playwright as a functional testing automation tool.

Testing software works best when there is a clear and organized approach. A research study [2] was conducted on different testing methods to create a software testing methodology that follows this structured approach. They have showed that using an organized test processes can help improve software reliability and, specifically, makes it easier to find defects. Thus, using a testing models can improve the overall software quality and make sure that the software runs as it is expected to.

Building on structured testing, the study conducted [3] to investigate the role of testing professionals in unit testing within Agile environments. They deployed survey-based research and collaborated with developers to strengthen unit testing practices. The research finding shows that integrating testers early improves software quality and fosters mutual learning between developers and testers.

In a research study [4], they proposed a technique for software maintenance that uses test classification, deletion, and selection algorithms. Using this proposed method, the study found that the test suite size was reduced by about 28%, which worked better than random selection or prioritization methods. Thus, the finding shows that organizing tests with a clear test method can make testing faster and more efficient without losing the product quality.

A research study was conducted [5] and explored the Agile testing processes and its challenges. They conducted literature reviews and industry analysis to understand testing integration in Agile. In conclusion, the study shows that a testing should be iterative and aligned with development. And also supported by a proper technique to make sure of the product quality.

In Agile methodologies, using Scrum has shown to be effective in a system development. In the research study [6], they implemented a Scrum methodology in developing the Warehouse Receipt System. With this, they have conducted system development and performed User Acceptance Testing, which achieved a 100% test results. The research findings shows that Scrum methodology can successfully implement and meet the user requirements.

According to research [7], they had comparison of effort estimation models including Cobb-Douglas, Neuro Fuzzy, and Genetic algorithms. They applied these models to software project estimation tasks. The research study concluded that Neuro Fuzzy has the highest accuracy for effort estimation, thenfollowed by Genetic and Cobb-Douglas methods.

A structured planning enhances a task management. A recent research study [8] proposed integrating the Work Breakdown Structure (WBS) with task completion forecasting by using a linear regression. With this, the researchers analysed the construction project data to improve time

and cost management. Overall, the study findings show that the WBS approach is effective that it can predicts task completion with low error and high explanation percentage.

Selenium has been around for a long time and is one of the main tools people use to automate web apps. [9] investigated at what happens when you use Selenium WebDriver with JUnit and ATF, and they found it can basically run tests and generate reports on its own. In this setup it doesn't just save time, but it also makes the tests more reliable, with fewer failures, and even uses memory better than the old manual ways of testing.

A recent research study [10] explored a VS Code extension that uses Selenium to simplify sending code to a WebGPU cloud platform. With this system, students can write code locally and then push it to the cloud with just a one click. Overall, the findings shown that this setup has cut down the manual work and made the whole process smooth. It also proved that Selenium could connect local coding environments with cloud platforms which help deployments happen faster and with a fewer problem.

Despite the popularity of Selenium as favoured automation tool, there were new tools has been released. In a research study [11], they compared Selenium and Playwright for automating employee management applications. Automation scripts were written in Python and timed how long each tool took to complete tasks. The study found that Playwright has handled the tasks in simplicity, since it finishes faster and made the whole process feel much more efficient than Selenium.

A recent research study [12] has developed a bug testing automation platform that uses Playwright with a backend API. And the platform was deployed in a cloud environment to make it handle the testing and also automated reporting. With this experiment, the research study found that connecting Playwright with backend API has made the system more scalable and easier to access. In addition, it has showed that this setup can really make testing more reliable and less hassle.

In research study [13] they have studied the software testing strategies and life cycles that can improve software quality. Also, they have reviewed the existing testing techniques and analysed its effectiveness, then suggested improvements. The study findings showed that having a clear testing plan and strategy do helps verify and validate software.

A study was conducted [14] to investigate challenges faced by most Agile testers in multinational IT organizations. On this study, they have collected information through a surveys, forums, and interviews. This collected information helps the study to understand common problems and best practices. Moreover, the findings showed that using a team management tool, also planning an effective sprint has reduced challenges in Agile testing.

Functional testing usually needs a multiple testing techniques just to make sure of a complete test coverage. With a recent book [15] it has reviewed methods such as the equivalence partitioning, boundary value analysis, and decision table testing, and that also analyzed how they are applied in real testing world practice. The study found it is important to use a technique because provides more quality testing which helps catch more potential issues in the software.

Regression testing can cost more time; thus, it works better when tests are organized and automated. A research study [16] has explored orchestration strategies for regression test suites. There were multiple testing techniques has been used to improve the regression testing. And the research study concluded that orchestrated strategies help choose, prioritize, and reduce tests. Thus, regression testing can be done faster and more efficiently in a real-world project.

In a research study [17] they proposed a TestSage for large-scale web service regression test selection. Also, they have collected function-level dependencies and make a test over a million functions at Google. The research findings show that TestSage has reduced the testing time up to 50% while at the same time maintaining the test accuracy, which has supported large-scale application testing.

Although many studies have examined automation tools such as Selenium, QTP, and UFT, most of these tools focus on standard web applications and do not fully address the complexities of EDI systems like Crossfire EDI. Previous research shows several limitations, but it also points to five key areas where this study can contribute. Firstly, there is an opportunity to explore how the system can be better prepared before an actual system update which is the post deployment checking of system readiness, make sure that it is stable first and can reduce the risk of unexpected issues. Secondly, there is potential to automate validation after deployment, confirming that the system can function correctly and that important processes has been unaffected. Thirdly, automating critical workflows and key features, which are necessary for the system's reliability. Fourthly, the test design and execution of consistent, reusable approaches for automation. Which can enhance testing efficiency and accuracy that addresses a gap in previous studies. Finally, reporting of results present an area where actionable insights for the developers can be provided to support a timely response and a decision making.

Overall, these points show how the study pushes beyond the limits of previous research, taking on challenges in testing the Crossfire EDI system that have not yet been fully explored. This work seizes the opportunity to deploy Playwright as an automated testing tool and implement it using C# within a .NET Framework environment, aiming to make automated testing more reliable, efficient, and impactful. By embracing these challenges, this research report seeks to deliver solutions that are practical, repeatable, and aligned with industry standards for EDI system testing.

3. RESEARCH METHODOLOGY

This study applied the Agile Scrum framework together with the Software Testing Life Cycle (STLC) to manage testing for the Crossfire EDI system. Scrum uses short, iterative cycles called sprints, where the team plans tests, designs them, executes, reviews results, and reflects on improvements. Moreover, STLC provides a clear, step-by-step process for testing, making sure nothing is overlooked. By combining Scrum with STLC, the team stayed flexible, adapted quickly to changes, and continuously improved testing practices.

Research supports [18] describe it as a practical and structured method to apply Agile principles. [19] explained that integrating testing into Scrum helps catch defects early and reduces the risk of breaking existing functionality. These studies show why Scrum is widely adopted in both academic research and industry projects.

Scrum was particularly suitable for this study because the Crossfire EDI system is complex and handles business-critical processes. Traditional testing approaches would have been slower and less adaptable. Using Scrum, the team could prioritize tasks based on risk and importance, work in manageable sprints, and get regular feedback from stakeholders. This approach kept testing focused, efficient, and aligned with project goals.

4. RESEARCH EXECUTION

The testing process for the Crossfire EDI system followed six defined phases. Namely, these are Requirement Analysis, Test Planning, Test Case Design, Test Environment Setup, Test Execution, and Test Closure. This is to structure the execution in alignment with the test lifecycle. Firstly, in requirement analysis, there is a meeting with developers to understand what the system needs, such as validating requirements for pre-deployment and postdeployment testing. Secondly, test planning was conducted by selecting Playwright for automation with the XUnit framework for execution. And also preparing a testing task schedule. Thirdly, test case designing is a phase to design the test cases based on the provided deployment checklist and best practices. Next, the test environment setup was prepared by installing Playwright, setting up local databases and a local website. Fifth, the test execution where all test cases areexecuted. Finally, analysing results based on the native reporting of Playwright with Xunit framework.

Figure 1: Software Testing Life Cycle for Crossfire EDI System

4.1.Test Plan for Crossfire EDI System

Test Plan

This test plan is designed to guide testing activities for the Crossfire EDI system. Its goal is to make sure that all major functions are working correctly before and after deployment, and that test results are clearly recorded and easy to compare.

Test Items

- Pre-deployment check automation scripts
- Post-deployment validation automation scripts
- Playwright-based execution and logging framework
- Automation Test Reporting

Features to be Tested

1.Pre-Deployment Checks:

- Capture list of the errored messages and interchanges
- Check Max Pool Size and Encrypt parameters in connection strings
- .NET Standard/Core installation verification

2. **Post-Deployment Checks:**

- Crossfire Admin: General/Advanced tabs, partner creation, events creation, messages, processes, functions, code lists, version updates, editor load verification, password changes
- Top Navbar Tabs: Dashboard, system tests, bulk processing
- Messaging: WorkWith list, server/transport/route/certificate creation/update

Features Not to be Tested

- Non-critical modules unrelated to deployment
- Exploratory testing outside pre- and post-deployment tasks

Approach

- Automation using Playwright with XUnit framework and C# language
- Modularization of test scripts using Page Object Model
- Pre-deployment screenshot data captured for comparison

Automated screenshots, logs, and test reports for verification

Item Pass/Fail Criteria

- **Pass:**All checks complete without errors and expected behavior confirmed
- **Fail:** Any error in functional workflow, logs, or configuration parameters

Suspension Criteria and Resumption Requirements

- If pre-deployment checks fail, deployment is suspended until issues are resolved.
- Failed post-deployment checks pause regression execution until fixed.

Test Deliverables

- Automated test scripts
- Test execution reports
- Defect reports

Testing Type

- **Functional Testing:** Verify that all system functionalities work as expected post-deployment.

Automation Testing Tool

- **Playwright** – chosen for its support of modern web applications, speed, and .NET compatibility

Test Design Approach

- Test cases are derived from pre- and post-deployment checklists.
- Critical functional paths are prioritized for automation.
- Logging, screenshots, and report generation are automated for traceability.**Work**

Breakdown Structure

Table 1: Test Estimation using Working Breakdown Structure (WBS)

Phases	Features	Task Description	Estimated Hours
1. Test Planning	All Testable Requirements	Review company requirements	8
	Literature Review & Requirements	Review existing studies and gather requirements	12
2. Test Design	Pre-Deployment Checks	Check exception logs for errors	2
		Capture screenshots of errored messages, interchanges, and failed messages	2
		Verify Max Pool Size and Encrypt parameters	1
		Ensure .NET Standard/Core is installed	1
	Post-Deployment Checks – Crossfire Admin	Verify General tab loading; create/update partners, events, messages, processes, functions, and code lists	4
		Verify Advanced tab loading; update module versions; verify TinyMCE& ACE editor	3

		Verify user password changes and production favicon	2
	Top Navbar Tabs	Verify dashboard tab, run system & function tests	3
	Bulk Processing	Verify bulk processing tab; run Bulk Export and Bulk Processing	2
	Messaging	Check WorkWithlists; create/update Servers, Transport Types, Transports, Routes, Certificates, and Schedules	6
	Engine & Load Balancer	Check logs, status, dashboard; verify load balancer	5
	FTP / Serv-U	Check FTP files and logs for missing users	3
	Monitoring & Add-ins	Ensure monitors are green; check add-ins including Excel transport	4
	Reporting & Logging	Capture screenshots, logs, and generate automated testing reports	3
3. Test Environment Setup	Environment Configuration	Setup testing environment for EDI system	5
	Constraints	Setup database, schemas, and centralized management	8
4. Test Execution	Pre-Deployment & Post-Deployment	Execute all test cases for pre- and post-deployment validation	12
	Performance & Usability Testing	Run performance and usability tests	10
5. Defect Management	All Features	Log, track, and retest defects found during test execution	10
6. Test Closure & Report Preparation	Final Regression & Reporting	Regression testing, finalize report and analysis	40

Environmental Needs

- OS: Windows 10/11
- Browser: Chromium
- Visual Studio Professional, .NET framework, XUnit
- Crossfire EDI system and database access

Test Execution

- Pre-Deployment: Capture errored messages, interchanges, and

Pos-Deployment: Execute scripts for Crossfire Admin tabs, Top Navbar, and Messaging.

4.2. Test Cases

The test case is a detailed instruction that guides how to go step by step to create the test. It helps to create test scripts that align with the expected results, which are based on the business requirements. The test cases were created using the provided checklist for both pre-deployment and post-deployment. Each requirement was reviewed and communicated to make sure the correct test data and expected results were gathered. Few of the test cases are shown below in Table 3.

Table 2: A test plan for the Crossfire EDI system

TC#	Test Case Scenario	Test Description	Test Steps	Test Data	Expected Result	Pass/Fail
TC_01	Pre-Deployment	To verify the presence of exception logs for any errors predeployment	1. Navigate to Crossfire Admin 2. Click General Tab 3. Screenshot exception logs exists	URL: http://localhost/	All exception logs are checked and known errors identified	PASS
TC_02	Pre-Deployment	To verify the list of Errored Messages and save an evidence to test screenshot folder	1. Navigate to Crossfire Admin 2. Click General Tab 3. Click Errored Mess+[ages 4. Take a screenshot	Logged in based on last user session state	Screenshot saved for compariso n post-deployment	PASS

TC_03	Pre-Deployment	To verify list of Check Errored Interchanges	1. Navigate to Crossfire Admin 2. Click General Tab 3. Click Errored Interchanges 4. Take a screenshot	Logged in based on last user session state	Screenshot saved for comparison post-deployment	PASS
TC_04	Pre-Deployment	To verify list of Check Failed Messages	1. Navigate to Crossfire Admin 2. Click General Tab 3. Click Failed Messages 4. Take a screenshot	Logged in based on last user session state	Screenshot saved for comparison post-deployment	PASS
TC_05	Pre-Deployment	To verify list of Check Max Pool Size & Encrypt parameters	1. Navigate to Crossfire Admin 2. Click Advanced tab 3. Click External	Logged in based on last user session state	Max Pool Size and Encrypt parameters verified	PASS
			Connections 4. Assert that			
TC_06	Pre-Deployment	To verify list of Max Pool Size	1. Click on Toll Connection 2. Validate Max Pool Size 3. Click Cancel	Logged in based on last user session state	Max Pool Size = 2048 verified for Toll, Middleware, Message, EDI_Cloudconnectic ns	PASS
TC_07	Pre-Deployment	To verify that .NET Standard/Core is installed	1. Click Advanced Tab → Assembly References 2. Verify .NET Standard/Core and required DLLs	Logged in based on last user session state	All required components are installed	PASS

TC_08	Crossfire Admin Navbar	To verify that the Dashboard tab loads correctly	1. Click Dashboard tab 2. Verify page fully loaded 3. Export page load counts	URL: http://localhost/	Dashboard page loads successfully; counts exported	PASS
TC_09	Crossfire Admin Navbar	To verify that the system is able to run a system test to validate process execution	1. Click System Test 2. Fill out fields 3. Tick checkbox 4. Run Test 5. Verify results	Process: MBA	System test completes successfully; results displayed	PASS
TC_10	Crossfire Admin Navbar	To verify that the system can run a function test	1. Click Function Test 2. Click a function 3. Run the test	Logged in based on last user session state	Systems runs the functions and shows test result	PASS

4.3. Solution Structure

The Crossfire EDI system needs a proper solution structure because it is connected with the real system codebase. To make it organized, the framework used was the Page Object Model (POM). This model separates the all the test data, the page selectors, and the test logic. Thus, the tests become more organized, easier to maintain, and most especially, the test structure can be understood by the next person.

- Authentication: This is a global login setup that is created so that all created tests will use the same authentication. This it to make the login process as one time process and be applied to all tests to avoid repetitive scenarios.
- Collection Fixture: The fixture gives a shared context for all tests. This helps to keep the code clean and also makes it easier to reuse.
- Test Data: The test data is stored in a dedicated test folder to compile it. This simply makes the test data easy to find, update, and manage. Thus, making it separate from the test data and pages.
- Test Pages: All page elements, selectors, and UI logic are saved in the Test Pages folder. This separation makes the framework more flexible and easier to update.
- Tests: The test folder contains the real test scripts. These scripts include the steps and checks that Playwright uses to run the tests.

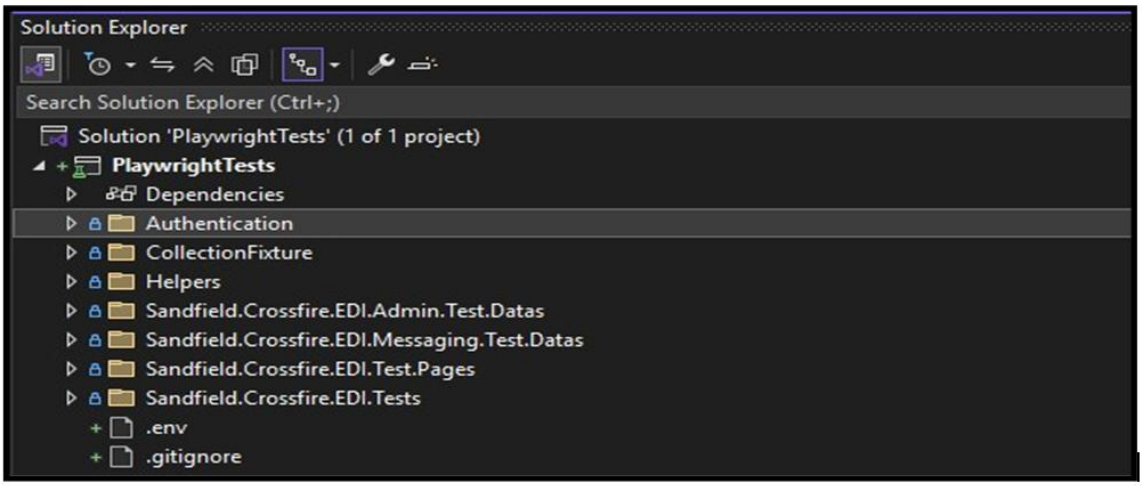

Figure 2: Page Object Model

Figure 2 shows the folder structure created for the Crossfire EDI system. The structure clearly indicates the purpose of each folder, providing a solid foundation for maintaining clean and organized scripts. This organization makes the scripts reusable and reliable for future testing needs.

4.4. Test Script Writing

Test scripts are pieces of code that instruct the system on how a test case should run. The first technique applied was to make each test independent. The Crossfire EDI system is a chain of related entities, where tests could potentially rely on each other. However, if one partner or event fails, all dependent tests could also fail, making it difficult to determine the root cause. To avoid this, each test case is designed to be separated from the others. No test reads or writes data that another test depends on. This makes sure that if one test fails, it does not cause other tests to fail, making it easier to identify the root cause of issues.

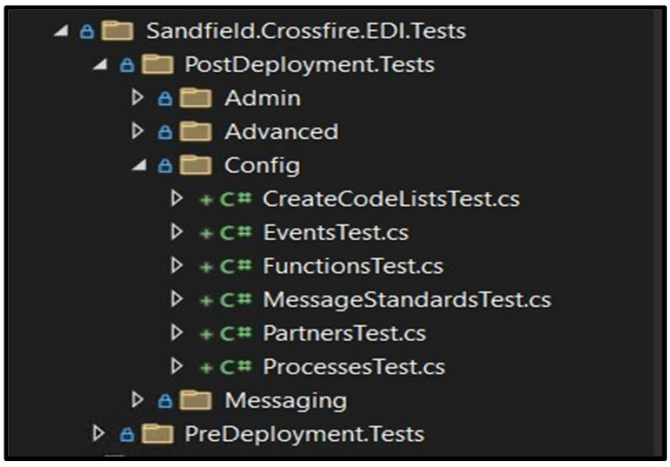

Figure 3 Test Classes for Crossfire EDI System

4.5.Test Data Handling

Handling test data is one of the most important parts of writing test scripts. The EDI system contains a large amount of data that can be reused for testing. However, if this data is deleted or changed in the future, tests that rely on it may fail. There are several ways to manage test data: it can be hardcoded or stored in external files (JSON, CSV, and more formats). This is particularly challenging for this system because automation deals with many different data types that need to be reused. For example, creating a partner requires an entity, and creating a transport also requires an entity.

```
using PlaywrightTests.Sandfield.Crossfire.EDI.Admin.Test.Datas.DataModels;

namespace PlaywrightTests.Sandfield.Crossfire.EDI.Admin.Test.Datas.Common
{
    14 references | 0 changes | 0 authors, 0 changes
    public static class CommonTestData
    {
        public const string EntityCommon = "Customer";
        public const string NameCommon = "This is an automated test Name";
        public const string PartnerCommon = "SAL - Automated Test";
        public const string GeneralCommon = "This is an automated test";
        public const string EventCommon = "This is an automated test event";


    }
}
```

Figure 4: Common Test Data

```
using PlaywrightTests.Sandfield.Crossfire.EDI.Admin.Test.Datas.Common;
using PlaywrightTests.Sandfield.Crossfire.EDI.Admin.Test.Datas.DataModels;

namespace PlaywrightTests.Helpers
{
    6 references | 0 changes | 0 authors, 0 changes
    public static class TestDataFactory
    {

        1 reference | 0 changes | 0 authors, 0 changes
        public static EventsData CreateNewEvent()
        {
            return new EventsData
            {
                Entity = CommonTestData.EntityCommon,
                Code = "TEST",
                General = CommonTestData.GeneralCommon,
            };

        }

        1 reference | 0 changes | 0 authors, 0 changes
        public static PartnersData CreateNewPartner()

        {
            return new PartnersData
            {
                Entity = CommonTestData.EntityCommon,
                Code = "TESTPD",
                Name = CommonTestData.NameCommon,
                Contact = "Test Contact"
            };
        }
```

Figure 5: Test Data Factory

To address this, a test data factory and common data file separation were created, as shown in Figures 4 and 5. Each test remains independent, but data can be shared across tests. This reduces repetition and makes scripts easier to maintain. The test data factory also organizes data in a way that each test can access exactly what it needs.

Login Authentication

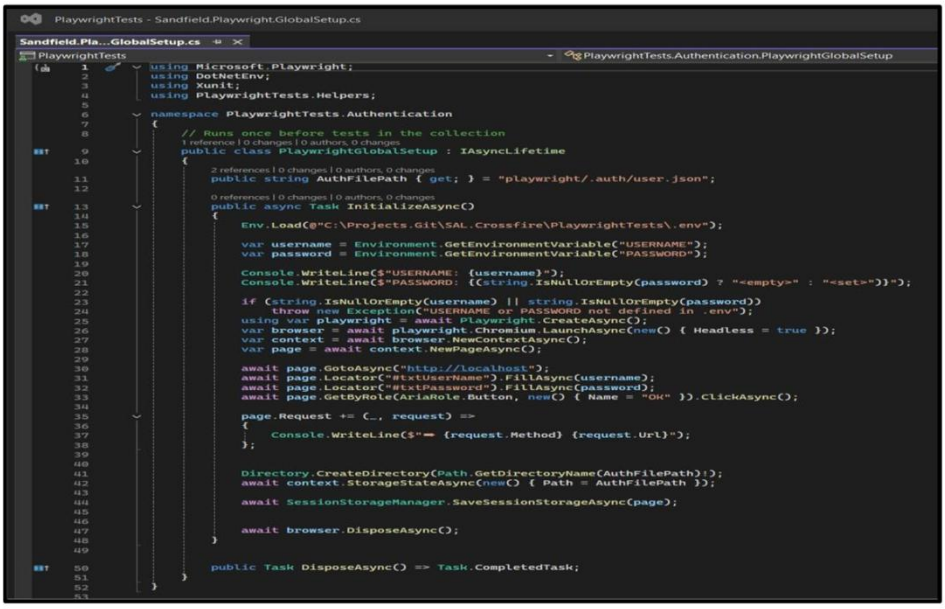

Figure 6: Global Setup for Login

In Figure 6, instead of having multiple logins for each test, the class PlaywrightGlobalSetup.csis used to manage authentication across all tests. This minimizes the need to log in every time a browser session starts. Because tests run independently, they normally start from the very beginning, including the login process. In addition, this script ensures that the username and password are stored in a .env file, avoiding the security risks of hardcoded credentials.

4.6.Test Class and Test Page

The [Fact] attribute in XUnitis used to define a single test, similar to the [Test] attribute in other testing frameworks. Figure 7 shows a test class for creating a new partner. The test script simply calls a task named CreatePartnerTest(), which performs all the necessary actions. These actions and the page selectors are implemented separately in a Page Class, as shown in Figure 8. This shows that main test classes can be written in a way that is easy to maintain and understand by the next person who is doing the test.

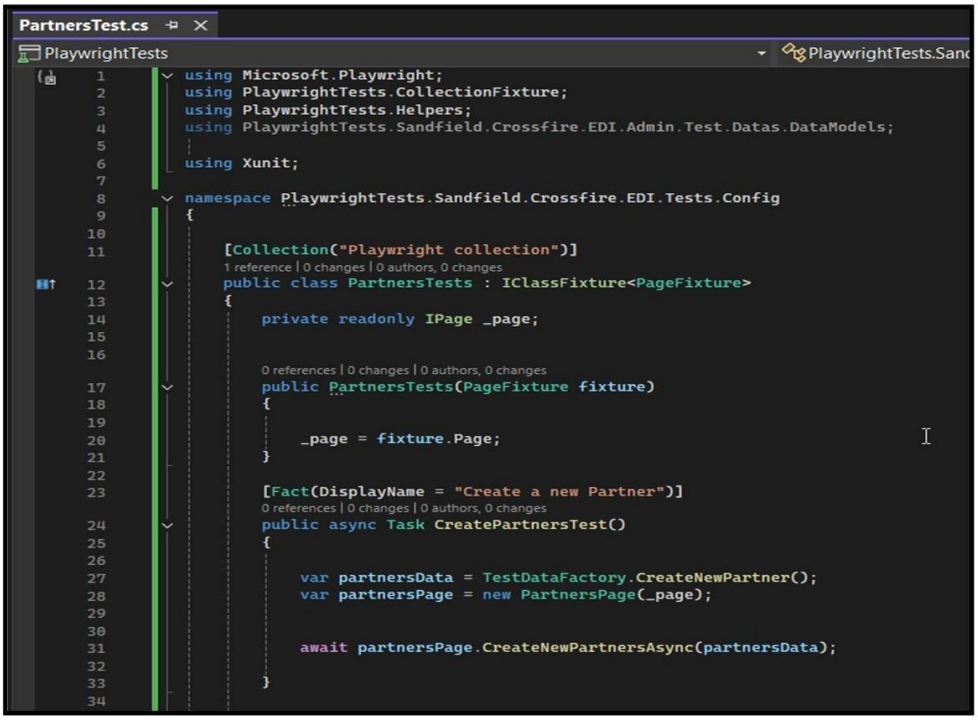

Figure 7: Test Class for Partner Creation

In Figures 7 and 8, it is shown how XUnit framework assertions were mainly used in writing the test scripts. Assertions like “await” and “async” has helped to reduce test flakiness, that usually happens when locators, pages, or data are not fully loaded before moving to the next test step. This is why the reliability of the tests has significantly improved. With XUnit framework, the system makes sure that each step of the test is completed properly before the next one starts. Although only the basic methods were applied in this research study, there are still many more features available in XUnit that can be explored for future testing.

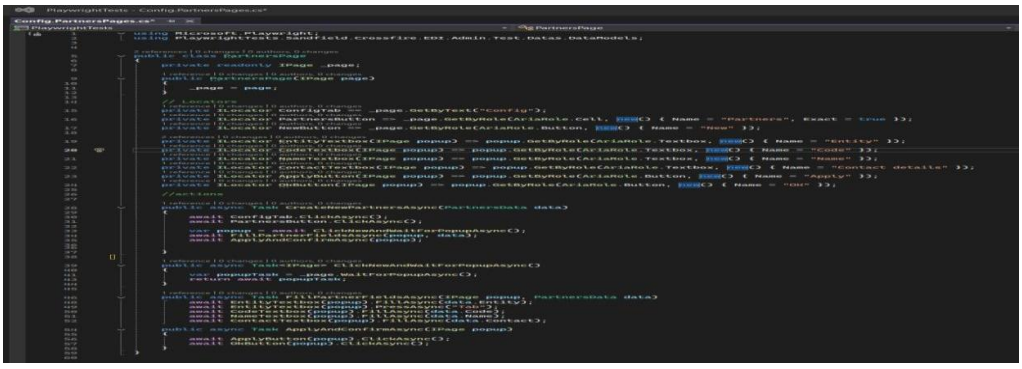

Figure 8: Test Page for Partner Creation

4.7.Headless vs Headed Test Execution

In headless mode, browsers do not open visually; instead, test scripts interact programmatically with the application, filling forms, clicking buttons, and validating outputs just like a user would. As shown in Figure 9, headless test execution for the Crossfire admin tabs took 23.5 seconds. By comparison, headed execution took 30.7 seconds, as shown in Figure 10. Headless mode is faster

because it does not render the browser UI, reducing the resources and time required to perform each action. Making Playwright headless execution meets Crossfire testing need of fast execution.

```
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.8.1+ce9211e970 (64-bit .NET 9.0.9)
[xUnit.net 00:00:00.11]   Discovering: PlaywrightTests
[xUnit.net 00:00:00.15]   Discovered:  PlaywrightTests
[xUnit.net 00:00:00.16]   Starting:    PlaywrightTests
USERNAME: Default1
PASSWORD: <set>
[xUnit.net 00:00:22.25]   Finished:    PlaywrightTests
  PlaywrightTests test succeeded (23.5s)

Test summary: total: 6, failed: 0, succeeded: 6, skipped: 0, duration: 23.5s
```

Figure 9: Headless Test Execution for Crossfire Admin

```
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.8.1+ce9211e970 (64-bit .NET 9.0
[xUnit.net 00:00:00.10]   Discovering: PlaywrightTests
[xUnit.net 00:00:00.14]   Discovered:  PlaywrightTests
[xUnit.net 00:00:00.14]   Starting:    PlaywrightTests
USERNAME: Default1
PASSWORD: <set>
[xUnit.net 00:00:29.49]   Finished:    PlaywrightTests
  PlaywrightTests test succeeded (30.7s)

Test summary: total: 6, failed: 0, succeeded: 6, skipped: 0, duration: 30.7s
```

Figure 10: Headed Test Execution for Crossfire Admin

4.8. Crossfire EDI System Test Execution

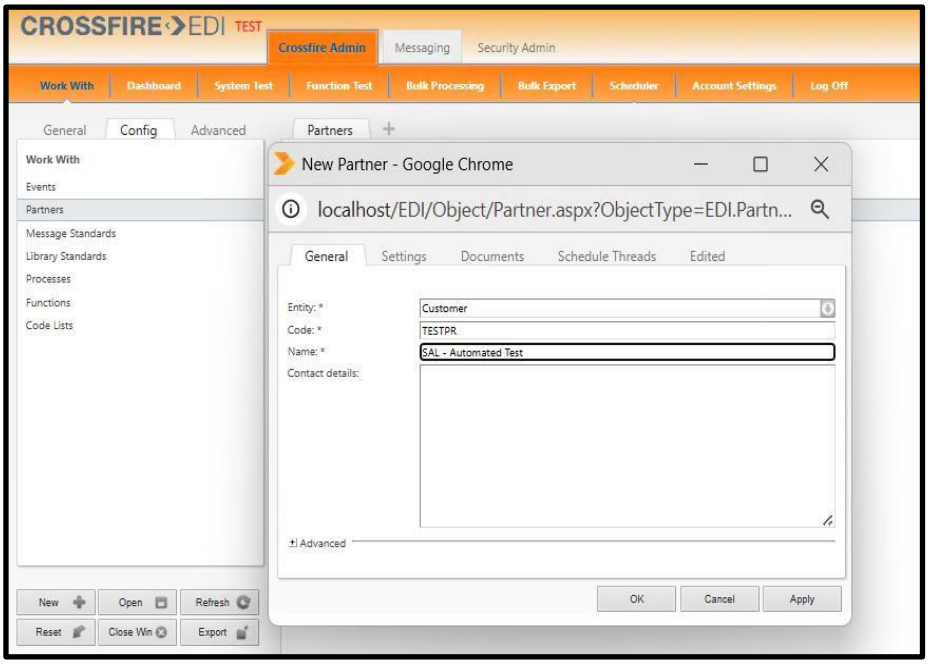

Figure 11: Headed test execution of post-deployment test

Figure 11 shows the Crossfire EDI system automating the partner creation process. In headed mode, the browser opens visually during test execution, allowing the tester to see the test running in real time. This helps verify that the script matches the intended test flow and makes debugging easier, as errors can be observed directly on the screen.

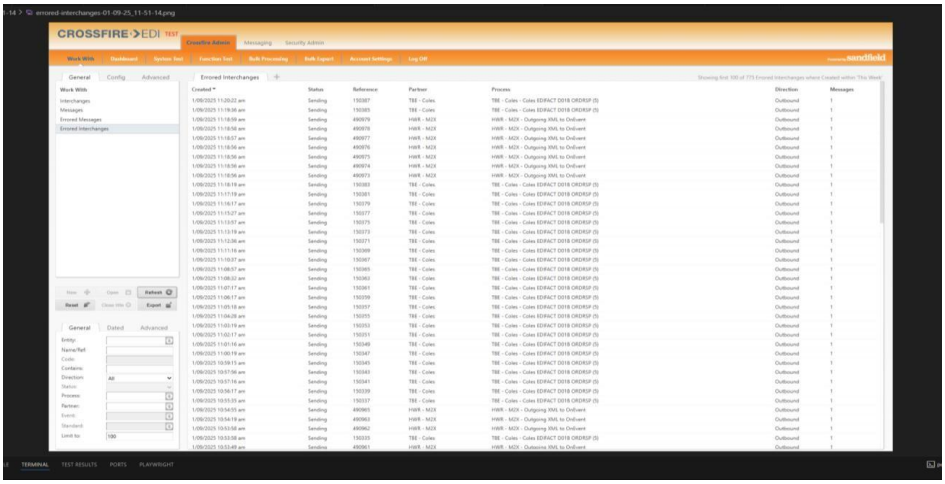

Figure 12: A screenshot of pre-deployment test

Figure 12 shows all errored interchanges captured during testing. Each failed message and its corresponding log file are automatically saved in the designated test folders. This gives developers and clients concrete evidence of the exact errors that occurred, including timestamps, partner IDs, and message details. By storing this information, the team can verify what went wrong, reproduce the issue if needed, and use it to improve or add new functionality in future.

5. RESULTS

This section shows the overall results from running the research's test cases. The goal is to see how well the system works under the planned tests and to give a clear view of the outcomes. The following discussion explains the key points from the test run.

5.1. Native Log Reporting

Playwright reporting has built reporting based on the framework and language chosen. Currently, the tests were conducted using the .NET framework with XUnit, and the results are presented through simple command-line reporting. As illustrated in Figure 6, all 33 test cases passed with no failures, indicating that the tested functionalities are performing as expected.

```
USERNAME: Default1
PASSWORD: <set>
[xUnit.net 00:00:00.11]   Discovering:    PlaywrightTests
[xUnit.net 00:00:00.15]   Discovered:    PlaywrightTests
[xUnit.net 00:02:18.50]   Finished:    PlaywrightTests
  PlaywrightTests test succeeded (138s)

Test summary: total: 33, failed: 0, succeeded: 33, skipped: 0, duration: 138s
PS C:\Projects.Git\SAL.Crossfire\PlaywrightTests>
```

Figure 13: Playwright summary test report

While this reporting approach is straightforward and functional, it may lack the advanced visualization and interactivity found in HTML-based or plugin reports using other language. However, it still serves the essential purpose of confirming that the test suite executed successfully and that no critical issues were encountered during the run. This simplicity also makes it easy to integrate into automated pipelines without additional configuration or tools. Most importantly, the result shows that the automated scripts have ran reliably that proves the research objective of replacing repetitive manual testing with an automation was achieved.

6. DISCUSSION

The purpose of this discussion is to interpret the results from the automated test execution, explain their significance, and show how the research met the target objectives of this research report. The research aimed to deploy Playwright for testing the EDI Crossfire System, which previously relied on manual testing.

6.1.Deploying Playwright as Testing Automation Tool

Playwright was chosen because it's a new tool that shows favourable results based on recent research. It compatible the modern browsers such as Chromium, Firefox and Webkit and at the same time it has multi-language and framework support. Setting it up was surprisingly fast as everything needed was included, so there was no long configuration phase. Such as installing drivers, reporting and other plugins. This meant more time could be spent actually writing tests, not taking more time over setup. Automating the repetitive work is a huge relief to traditional manual testing; instead of manually checking long list of items, the system now did test reliably in minutes.

6.2.Pre-Deployment Testing of Crossfire EDI System

Every system update brings a risk of errors, making the system pre-deployment testing a necessary. Traditionally, the team manually capture a screenshot of logs, error messages, failed interchanges, and key settings like Max Pool Size and encryption. While this is manually doable, the process was slow and could possibly miss a single step. Using Playwright, all these validation checklists were automated. The Playwright tool automatically captures a screenshot of errored list of messages/interchanges. The captured evidence is then stored in the automatically created directory, which compiles the test with the indicated test date and time, making it really helpful for the developers to have. Because this pre-deployment evidence is used to compare against the postdeployment evidence. Making debugging easier for the developers in times of defect or bug detected.

6.3.Post-Deployment Testing of Crossfire EDI System

Once the Crossfire EDI system is confirmed stable; the developers can implement their change to the system. Post-deployment testing had to make sure that new updates did not break any existing functionality, especially Crossfire Admin functions. In practice, this was frustrating because testing task requires the developer to interact to every component of the system. Also, manually following the long checklists was very repetitive and will take a lot of time to finish. With Playwright, the reusable scripts ran consistently across different platforms. Watching all the 33 test cases pass was a clear indicator that automation could reliably replace tedious manual checks.

6.4.Automating the critical functionalities of the system

Automating the main functionality such as creating partners, transports, schedules, messages, and processes was challenging because these are server-side changes rather than simple locator clicks. Writing scripts and structured code was challenging at first, but once the framework was in place using XUnit, errors were easily caught and detected, and it was easy to pinpoint exactly which test had failed since reports shows them. For this test, a local database is freshly migrated to local machine, where there is completely no data, thus there are no captured errors, although the test correctly automates the function of capturing the list. And suggestions for this will be on the recommendation section.

6.5.Reporting of Test Execution

The beauty of automation is getting to see the reporting result, and what test scenarios has passed or failed. Playwright has a built-in reporter. However, the test runners differ depending on language chosen because each language has different testing ecosystem. In this researchstudy .NET language is used with XUnit testing framework. The way it lays the report is straightforward as it shows how many tests has passed, failed, and skipped.In this study, two types of language were tested when studying the playwright. And it shows that Javascript has HTML reporting because it uses Node.jstest runner. However, with XUnit , the terminal command line is able to generate a very simple report that would straight away know how many tests succeed, failed, and skipped.

6.6.Implication in Other Areas of Study

While automation significantly improves efficiency and accuracy, it is not a complete replacement for a human wise judgment, The test scripts require maintenance, and some edge cases may still need manual verification. However, automation has significant contribution in improving the testing practice and this approach can be applied to other enterprise systems, such as banking software, ecommerce websites where repetitive functional testing is common.

6.7.Contribution to the Study

This study addresses a manual testing of the Crossfire EDI system. By deploying Playwright, the research study shows how modern automation tools can reliably cover complex workflows. Thus, reducing errors and saving time for testing the system. Also, it contributes knowledge on applying Playwright to EDI systems, which is less documented in current research.

6.8.Interpretation of the Study

The overall research findings show that automated testing using Playwright, it automates all the 33 test scenarios showing a complete test coverage. Also, it has demonstrated how powerful XUnit library is. Because it runs the Playwright scripts inside the .NET framework and checks the results with assertions, and at the same time generates a report whether each test passed or failed. Generally, this research study has automated the Crossfire EDI system that reduced the manual efforts of the Sandfield’s developers, which also minimizes the accidental human error when testing. Thus, making deployment more reliable that benefits both developers and clients and the organization.

7. CONCLUSION

This research began by understanding the testing practices for the Crossfire EDI system. Currently, the main problem is the routinely manual testing for pre- and post-deployment checklist validation. By understanding the pain points of the developers and how it affected their clients, we started to propose automation of this testing workloads. Seeing how the developers work with the long list of code, and then afterwards deal with the long list of testing is definitely a tiresome work. With that, the main research points were developed, which is to design and execute scripts using a new tool, Playwright, and then evaluate the system's ability to handle it. There are different automation tools out there, but following the recommendation based on recent research, playwright has shown favourable results. And thus, applied and hope it will work also on this research study. Automating testing does not only work make triple faster, but it makes testing practice more reliable. One practical reason is because all these tests are designed and written to follow instructions that a machine would certainly cover to run. This is important for the complex workflow of Crossdire EDI system. By automating each process, it is definitely a significant contribution in reducing manual testing.

Now, the problem of going through the long list of testing, which approximately would take hours to finish has reduced to minutes of "you do not have to do anything" testing. This shows that playwright powerfully increased the team's productivity in different important things to work on. This was achieved with a proper test planning and analysis of the requirements and following through the Software Testing Lifecycle (STLC).

Overall, by deploying Playwright, we achieved our target of automating all testing of the Crossfire EDI System functionalities. This not only solved the problem of manual effort and long testing times but also improved the reliability of the tests.

8. RECOMMENDATIONS

In this section, the following recommendations are provided based on the challenges and limitations encountered during the research execution. One significant challenge was incorporating negative testing due to large testing coverage and timeframe. Negative testing anticipates errors or unexpected inputs, making sure that the system behaves correctly under failure conditions. By including these scenarios, the automation suite goes beyond simply verifying positive flows where it actively catches potential failures before they impact operations. The XUnit framework's rich library of assertions and expectations provides the tools needed to implement these tests efficiently. Another challenge was maintaining a consistent and clean local database for tests. To address this, it is recommended seeding the database with test data before execution and cleaning it afterward. This method guarantees that each test runs independently, avoiding any domino effect of failures caused by leftover data. By implementing these, the team can improve test coverage, system reliability, and the overall efficiency of automated testing.

REFERENCES

[1] Karatas, C., &Gultekin, M. (2021). EDI based secure design pattern for logistic and supply chain. 2021 9th International Symposium on Digital Forensics and Security (ISDFS), 1–5. Elazig, Turkey.

[2] Lopuha, O., Tsiutsiura, S., Poplavskyi, O., Lysytsin, O., Bondar, O., & Kruk, P. (2023). Test design methodology for software verification. In 2023 IEEE International Conference on Smart Information Systems and Technologies (SIST) (pp. 241–245). IEEE.

[3] L., Campos, O., Santos, R., Magalhaes, C., Santos, I., & d. S. Santos, R. (2024). Elevating software quality in agile environments: The role of testing professionals in unit testing. In 2024 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 293–296). IEEE.

[4] Lawanna, A. (2016). Test case design based technique for the improvement of test case selection in software maintenance. In 2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE) (pp. 345–350). IEEE.

[5] Pathak, K., Ninoria, S., &Bharadwaj, S. (2022). Scope of agile approach for software testing process. In 2022 11th International Conference on System Modeling& Advancement in Research Trends (SMART) (pp. 1079–1083). IEEE.

[6] Suwarni, B. N., Indriyanto, E. R., Kaburuan, P., Parwito, E., Darwiyanto, E., &Simatupang, J. W. (2018). Implementation SCRUM method in warehouse receipt system development. In 2018 International Conference on Orange Technologies (ICOT) (pp. 1–5). IEEE.

[7] Jha, M., &Jha, R. (2020). Comparing the effort estimated by different models. In 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 1148–1154). IEEE.

[8] Kurniawati, A., &Wisena, S. (2023). Integrate WBS and forecast task completion in construction projects using data analytics. In 2023 International Conference on Technology, Engineering, and Computing Applications (ICTECA) (pp. 1–6). IEEE.

[9] Sawant, K., Tiwari, R., Vyas, S., Sharma, P., Anand, A., &Soni, S. (2021). Implementation of Selenium automation and report generation using Selenium WebDriver & ATF. In 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT) (pp. 1–6). IEEE.

[10] Bai, H. (2021). VSC-WebGPU: A Selenium-based VS Code Extension for local edit and cloud compilation on WebGPU. In 2021 IEEE 3rd International Conference on Frontiers Technology of Information and Computer (ICFTIC) (pp. 474–477). IEEE.

[11] Apriansah, M. R., &Desanti, R. I. (2024). Efficiency comparison of employee management automation using Selenium and Playwright: A case study of technology consulting company. In 2024 International Conference on Information Technology Systems and Innovation (ICITSI) (pp. 419–425). IEEE.

[12] Vadia, K., Thukrul, A., Mazumdar, P. G., Panda, N., &Sirsat, A. (2024). Bug testing automation with Playwright and a backend API. In 2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC) (pp. 1867–1870). IEEE.

[13] Joshi, S., &Kumari, I. (2022). Analyses of software testing approaches. In 2022 International Interdisciplinary Humanitarian Conference for Sustainability (IIHC) (pp. 1276–1281). IEEE.

[14] JeevaPadmini, K. V., Kankanamge, P. S., Bandara, H. M. N. D., &Perera, G. I. U. S. (2018). Challenges faced by agile testers: A case study. In 2018 Moratuwa Engineering Research Conference (MERCon) (pp. 431–436). IEEE.

[15] Homes, B. (2022). Testing techniques. In Advanced Testing of Systems-of-Systems, Volume 1: Theoretical Aspects (pp. 197–231). Wiley.

[16] Greca, R., Miranda, B., &Bertolino, A. (2023). Orchestration strategies for regression test suites. In 2023 IEEE/ACM International Conference on Automation of Software Test (AST) (pp. 163–167). IEEE.

[17] Zhong, H., Zhang, L., &Khurshid, S. (2019). TestSage: Regression test selection for large-scale web service testing. In 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST) (pp. 430–440). IEEE.

[18] Srivastava, A., Bhardwaj, S., &Saraswat, S. (2017, May). SCRUM model for agile methodology. In 2017 International Conference on Computing, Communication and Automation (ICCCA) (pp. 864-869). IEEE.

[19] Kaur, S., Hooda, S., &Deo, H. (2023). Software quality management by Agile testing. In Agile Software Development: Trends, Challenges and Applications (pp. 221–233).