ATCSJ
Applied Technology and Computing Science Journal

Open Access **p-ISSN 2621-4458 | e-ISSN 2621-4474**

Available online to journal2.unusa.ac.id

UNUSA

S4-Accredited – SK No.79/M/KPT/2023

Journal Page is available to https://journal2.unusa.ac.id/index.php/ATCSJ

Performance Comparison of Automated Website GUI Testing Tools: A Study of Selenium IDE, Katalon Studio, UI.Vision, and BugBug

Muhammad Javier Dafa Rozi[1], Endang Sulistiyani[2,*],Rizqi Putri Nourma Budiarti[3]

[1,2,3] *Information System, Universitas Nahdlatul Ulama Surabaya, Indonesia*

Jl. Raya Jemursari 51-57, Surabaya, Indonesia

[1]muhammadjavier048.if20@student.unusa.ac.id, [2*]sulistiyani.endang@unusa.ac.id,

[3] rizqi.putri.nb@unusa.ac.id

Article history:

Received 20 October 2025
Revised 10 November 2025
Accepted 28 November 2025
Available online 2 December 2025

Keywords:

Automated GUI Testing,
Software Testing Tools,
Selenium IDE,
Katalon Studio,
UI.Vision and BugBug.

Abstract

Software testing consumes approximately 50% of development time and cost. One of the most commonly used testing activities is GUI testing, which is still frequently carried out manually. In manual testing, testers often repeat the same actions multiple times, increasing the risk of human error. In contrast, automated testing utilizes specialized tools to minimize these errors. Therefore, selecting the right tool and understanding its performance are crucial. Previous studies on automated testing generally focused only on implementing test objects without evaluating the actual performance quality of the tools used. There are many automated testing tools available that offer capture-and-replay features, such as Selenium, Katalon Studio, UI.Vision, and BugBug. This study compares the performance of these tools for website GUI testing using two parameters: execution time and encountered issues. A total of 21 test cases were designed and executed by the researcher. The execution results show that Katalon Studio, UI.Vision, and BugBug successfully completed all test cases as expected. However, Selenium failed to execute 5 test cases due to its inability to perform hover dropdown actions. In contrast, Katalon Studio, UI.Vision, and BugBug did not encounter issues during execution. In terms of test execution time, Katalon Studio recorded the fastest average time at 6.15 seconds, followed by Selenium. UI.Vision and BugBug required longer execution times, with average times of 20.85 seconds and 22.8 seconds respectively.

[2*] Corresponding author

ISSN 2621-4474 (online) 2621-4458 (print) © 2025 The Authors. Published by UNUSA
This is an open access article and licensed under a Creative Commons Attribution-Share Alike 4.0 International License
DOI: https://doi.org/ 10.33086/atcsj.v8i1.8368.

I. INTRODUCTION

Software testing has become a crucial aspect of modern software development, where approximately **50% of development time and cost** is consumed by testing activities [1]. As stated by Setiawan *et al.* [2], the software testing phase plays a vital role in ensuring the quality and reliability of a software product. Within the software development life cycle, testing represents the final stage prior to the release of the product to end users, making this stage essential for validating the system's readiness [3].

The primary objective of software testing is to identify and correct errors, deficiencies, or unmet requirements so that the final product achieves the expected quality standards and user expectations [4]. Testing can be performed either manually or automatically through the use of software testing tools. In manual testing, testers interact directly with the system to evaluate its functionality [5]. However, manual testing often involves repetitive actions, which can lead to human error and inefficiencies.

Conversely, automated testing minimizes human error by automating repetitive tasks through specialized tools. Automated software testing has therefore become an integral component in modern application development, enabling more scalable and consistent validation processes [6].

Among the various testing activities, unit testing, integration testing, and Graphical User Interface (GUI) testing are the most commonly executed [7], [8]. GUI testing is still frequently performed manually, where testers click through interface components based on predefined test scenarios [8]. Manual GUI testing poses several limitations, such as repetitive actions, incomplete coverage of interface elements, difficulties in failure detection due to insufficient command tracking, and the absence of automated test-time recording [9].

GUI serves as a medium of interaction between users and the underlying system or software application [10]. As such, GUI functionality significantly influences a system's overall quality. One critical issue in GUI testing concerns execution time, which is affected by interface design characteristics. Faster response times enhance user comfort and experience [9].

One technique frequently used in automated GUI testing is **Capture and Replay**, a black-box approach that records user actions within the application's interface and automatically converts them into reusable test scripts [11]. The rise of this technique has contributed to the widespread use of automated GUI testing tools.

However, the availability of numerous automation tools presents challenges in choosing the most appropriate one, assessing execution efficiency, and understanding the reliability of produced test results [7]. Many previous studies have focused on using automation tools solely for testing the functionality of the target application without analyzing the tool's own performance characteristics [12], [13]. Moreover, several comparative studies have evaluated only one to three tools, which limits the comprehensiveness of tool selection insights [14], [15].

A variety of automated GUI testing tools that support capture-and-replay functionalities exist, including Selenium IDE, Katalon Studio, UI.Vision, and BugBug. In this study, the authors compare the performance of these four tools. Selenium IDE and Katalon Studio are frequently used automation tools in academic research [16], while UI.Vision and BugBug are less commonly examined in the literature.

This study evaluates the performance of these tools in automated GUI testing on a website, using two evaluation parameters:

1. Execution time, and
2. Testing constraints or limitations encountered by each tool.

The GUI testing is conducted on the Esorogan UNUSA website, selected because it has low application complexity, defined as having fewer than 20 functional features [7]. In addition, the website is relevant to

the researchers' academic environment, and the testing results will contribute to enhancing system quality within that context. The objective of this research is to understand the efficiency and performance differences among various automated GUI testing tools, thereby supporting improvements in software testing quality.

II. LITERATURE REVIEW

This section presents previous research relevant to the problem addressed in this study. The first related study is titled "Kajian Otomatisasi Pengujian GUI: Selenium IDE, UiPath Studio, Katalon Studio" conducted by Hasanah [7]. This study compared three automated testing tools and found that Katalon Studio demonstrated superior performance in terms of minimizing testing constraints compared to the other two tools. Furthermore, the study showed that Selenium had the fastest learning time, as users did not require advanced scripting knowledge to perform tests. The difference between the previous study and the present research lies in the methodology; the earlier study incorporated an additional BDD (Behavior-Driven Development) approach, whereas the current study expands the comparison by adding more testing tools.

The second related study, titled "A Comparative Study of Automation Testing Tools for Web Applications" by Pelivani and Cico [14], examined automated testing tools applied to web-based applications. The study reported that Katalon Studio exhibited longer execution times than Selenium, primarily due to differences in programming languages; Katalon uses Groovy, which is more complex than the programming language used in Selenium. The difference from the present study is that the earlier research compared only two tools, while this study includes four automated testing tools, providing a more comprehensive comparison.

The third study, "Overview and Analysis of Automated Testing Tools: Ranorex, TestComplete, Selenium", compared three automated testing tools and found that Ranorex is well-suited for web-based applications due to its extensive built-in features and the fact that it does not require scripting knowledge. Selenium, being open source, offers advantages in reducing costs and is compatible across platforms and browsers, with strong reporting capabilities. TestComplete provides an easy-to-learn UI and efficient playback performance, making it suitable for applications with lower security requirements. The study also noted that beginners are encouraged to use Ranorex due to its simplicity and its scripting-free testing environment. While this prior research compared automated testing tools, the main difference from the present work is the evaluation parameters. The previous study used nine comparison criteria, such as speed, cross-platform capability, scripting language, data-based testing, playback abilities, application support, ease of learning, cost, and reporting. Whereas the current research focuses on two key parameters: testing constraints and execution time [17].

The fourth study, "Comparative Analysis of Automated Testing Tools on GUI Web-Based Applications" by Melia and Putra [10], is directly related to the present research as it compares automated testing tools using a web-based application as the object. The study concluded that Katalon Studio excels in ease of installation, interface usability, and the availability of built-in features, while Selenium WebDriver provides faster test-case execution speeds. The difference from the current research is that the earlier study compared only two tools, whereas this study evaluates four automation tools, offering broader analytical coverage.The fifth related work, "Comparative Study of Automated Testing Tools for Android" by Barus and Leo [18], examined automated testing tools for Android applications.

The study utilized 14 evaluation criteria, finding that Selendroid met the highest number of criteria overall. However, when evaluating the two primary criteria such as execution time and memory usage. The results showed that UI Automator offered the fastest execution time, while Calabash required the least memory. The difference from the current study is that the previous research focused on Android-based applications, whereas the present work evaluates web-based GUI applications.

III. METHODS

The stages of this research were carried out based on the research methodology diagram presented below.

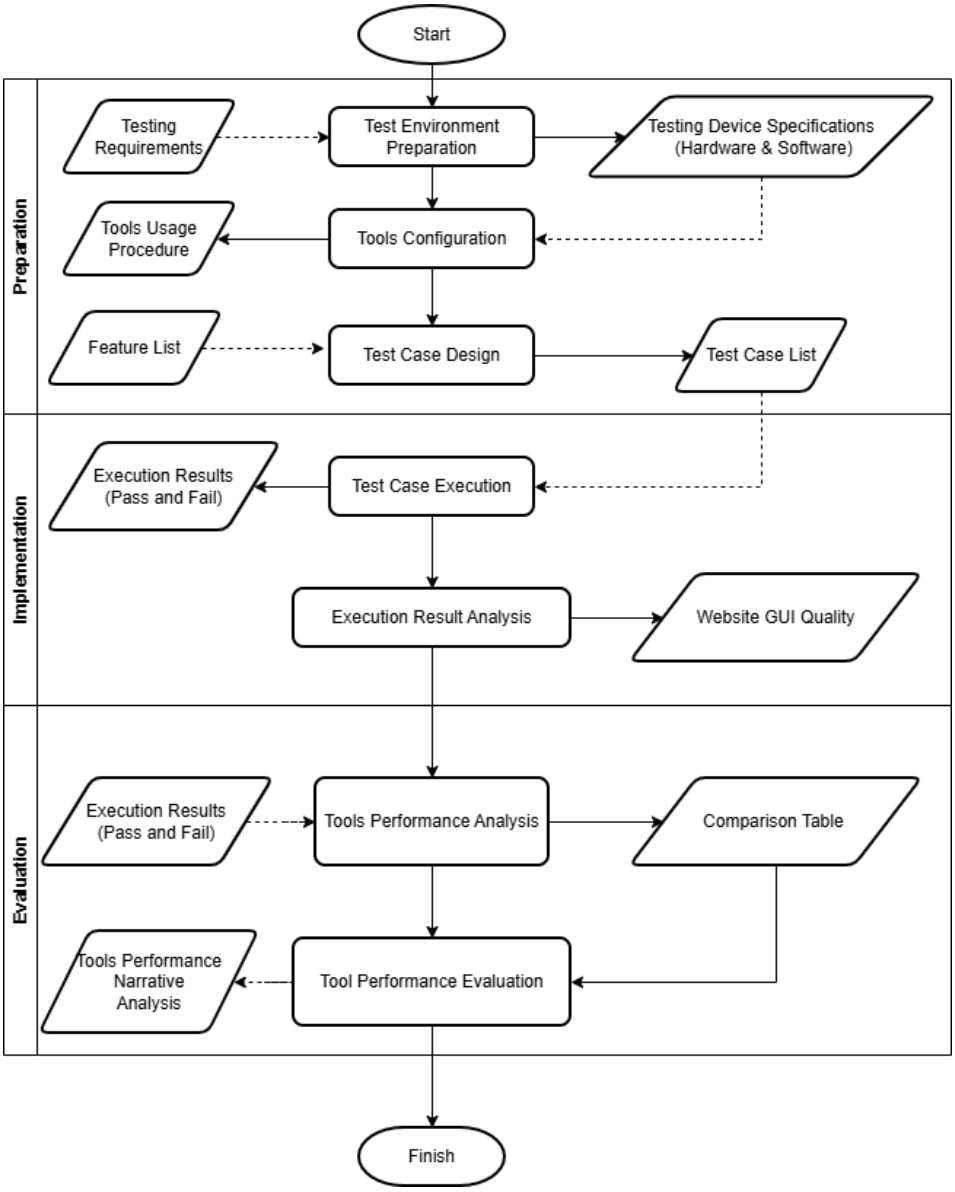

Figure 1. Research Methodology

A. Preparation Stage

The first stage in completing this final project is the preparation stage, during which the author conducted several processes, including the preparation of the testing environment, tool configuration, and test case design. Each process is described as follows:

1.1 Preparation of the Testing Environment

In this stage, the author performed an analysis of the testing requirements. The purpose of this process is to determine the necessary specifications of the testing environment to ensure that testing activities can be carried out effectively and accurately. The process begins with the identification of testing needs, followed by selecting the appropriate hardware, determining the suitable operating system, and specifying the software tools and objects to be used. The testing tools used in this study are Selenium IDE, Katalon, UI.Vision, and BugBug**.** The object being tested is the Esorogan UNUSA website**,** specifically from the roles of *Student* and *Lecturer*, as those are the only roles available on the system. This website is part of the author's academic environment, and testing it contributes to improving the quality of systems used within the institution. The output of this stage is a specification of the testing environment, detailing the hardware and software used throughout the testing process.

1.2 Tool Configuration

The purpose of this stage is to prepare and configure the testing tools that will be used during the testing process. The input for this stage consists of the selected tools based on testing requirements. This stage involves installing and configuring each tool to match the testing environment and ensure accurate results. The output of this process is a set of procedures for using the tools, including step-by-step guidelines on how to operate and utilize them effectively. This stage ensures that the testing tools can be used efficiently and correctly.

1.3 Test Case Design

During this stage, the author designed test cases based on the features of the object under test. The purpose of this stage is to support the testing process by developing detailed testing scenarios. The input for this stage consists of a list of features from the system being tested. The process includes identifying testing conditions, determining the necessary steps, and specifying the expected outcomes for each test case. The output is a comprehensive list of test cases that serves as a guide for validating the software and ensuring the desired level of performance and quality.

The test case table typically consists of the following columns:

- Feature: the software component being tested (e.g., login, registration).
- Test ID: a unique code used to identify a specific test case.
- Test Case Name: a brief description of what is being tested.
- Description: a detailed explanation of the purpose and scope of the test case.
- Type: the category of the test (e.g., positive or negative).
- Scenario: the steps performed during testing.
- Expected and Actual Results: including a final conclusion of *"Pass"* or *"Fail."*

The table serves as a structured reference for conducting and tracking the testing results.

B. Implementation Stage

The next stage is the implementation stage, which consists of test case execution and analysis of execution results. Each process is described below:

2.1 Test Case Execution

This stage involves executing the designed test cases. The purpose of this process is to evaluate the functionality and quality of the software to ensure that it operates as expected. The input for this stage is the complete set of test cases. Execution is carried out systematically, with each test case being performed according to its defined steps. The testing tools selected during the preparation stage are used to support this process. The output of this stage is the test execution results, categorized as *"Pass"* or *"Fail."* These results reflect the quality and correctness of the software.

2.2 Analysis of Execution Results

This stage aims to analyze the results of the executed test cases to provide deeper insights into the software's quality and to identify weaknesses discovered during testing. The input consists of the test execution results. The analysis process includes comparing actual results with expected outcomes. If a test case fails, the author reviews the cause of failure and traces the steps involved. This stage also evaluates software reliability. The output is a detailed understanding of the software quality and identification of areas requiring improvement.

C. Evaluation Stage

The final stage of this project is the evaluation stage, which consists of analyzing tool performance and evaluating tool performance.

3.1 Analysis of Tool Performance

The purpose of this process is to understand the performance level of each testing tool used. The input for this stage is the test execution results. We prepare two comparison tables:

1. Testing Constraints Table – documenting obstacles encountered during test case execution for each tool.
2. Testing Speed Table – recording the execution time for each test case on each tool. After collecting the execution times, the author calculates the average execution time using the formula:

Average Testing Time = (Total Execution Time) / (Number of Test Cases)

The outputs include the constraint comparison table and the execution time comparison table, both of which help evaluate each tool's strengths and limitations.

3.2 Evaluation of Tool Performance

Based on the performance analysis, the author evaluates each testing tool. The purpose of this process is to provide an overall assessment of the tools used during test execution. The evaluation considers testing constraints and execution time to determine the efficiency and reliability of each tool.

The output of this stage is a narrative evaluation describing the performance, reliability, and efficiency of each testing tool.

IV. RESULTS AND DISCUSSIONS

A. Preparation of the Testing Environment

The first step carried out by the author was preparing the testing environment, which required both hardware and software specifications. The hardware specifications used in this study are as follows:

1. Processor**:** AMD Ryzen 5 4600H with Radeon Graphics (12 CPUs), 3.0 GHz
2. Graphics Card**:** NVIDIA GeForce GTX 1650
3. RAM**:** 8 GB

The software specifications required for the research include:

1. Windows 11 Home Single Language 64-bit
2. Selenium IDE Extension
3. Katalon Extension
4. UI.Vision Extension
5. BugBug Extension
6. Google Chrome Browser
7. Mozilla Firefox Browser
8. Esorogan UNUSA Website

B. Tool Configuration

In this stage, we configured the tools that would be used during the testing process. Four tools were utilized—Selenium, Katalon, UI.Vision, and BugBug. Before executing the test cases, we needed to understand the workflow of each tool. Therefore, a testing procedure for each tool was prepared.

- Selenium

Table 1. Selenium Procedure

No.	Procedure
1	Add the Selenium extension to your browser.
2	Open the Selenium IDE page in the browser where the extension is installed.

3	Select **"Create a new project"**, enter the desired project name, and click **"OK"**.
4	Click the **"+"** icon to add a new test case, enter the test case name, and click **"Add"**.
5	Click the **"REC"** button at the top-right corner or press **CTRL + U**, type the target website URL, and click **"START RECORDING"** to capture interactions on the website.
6	Click the **"REC"** button again at the top-right corner of Selenium IDE to save the recorded test script.
7	Press **CTRL + R** to replay the recorded interactions on the website.
8	Wait until the replay process is complete.

- Katalon

Table 2. Katalon Procedure

No.	Procedure
1	Add the Katalon Recorder extension to your browser.
2	Open the Katalon Recorder page in the browser where the extension is installed.
3	On the left, in the workspace column, click the **"+"** button under **Test Suites** to add a new test suite. Enter the desired test suite name and click **"OK"**. A new test suite will appear.
4	Click the **"+"** button on the newly created test suite to add a test case. Enter the desired test case name and click **"OK"**.
5	Click the **"REC"** button at the top-right corner or press **CTRL + U**, type the target website URL, and click **"START RECORDING"** to capture interactions on the website.
6	Click the **"Record"** button to capture interactions on the website.
7	Return to the Katalon page and click **"Stop"** to save the recorded test script.
8	Click **"Play Test Case"** to replay the recorded interactions on the website.
9	Wait until the replay process is complete.

- UI.Vision

Table 3. UI.Vision Procedure

No.	Procedure
1	Add the UI.Vision extension to your browser.

2	Before accessing the UI.Vision page, open the target website to be tested.
3	Open the UI.Vision page in the browser where the extension is installed.
4	Click the folder icon next to the **"+Macro"** button to create a new folder. Enter the desired project name and click **"Create"**.
5	Right-click on the newly created folder and select **"New Macro"** to add a test case.
6	Click the **"Record"** button to capture interactions on the website.
7	Return to the UI.Vision page and click **"Stop Record"** to save the recorded test script.
8	Click **"Play Macro"** to replay the recorded interactions on the website.
9	Wait until the replay process is complete.

- BugBug

Table 4. BugBug Procedure

No.	Procedure
1	Add the BugBug extension to your browser.
2	Open the BugBug page in the browser where the extension is installed.
3	Click **"New Project"** to create a new project. Enter the project name and target website, then click **"Create Project"**.
4	On the left menu, select **Test** and click **"New Test"** to add a new test case. Enter the test name and click **"Create Test"**.
5	Enter the target website URL and click **"Start Recording"** to capture interactions on the website.
6	On the right side of the page, a BugBug popup appears. Click **"Finish and Close"** to save the recorded test script.
7	Click **"Run"** to replay the recorded interactions on the website.
8	Wait until the replay process is complete.

C. Test Case Design

In this stage, the author designed test cases based on the available features in the system. A total of 21 test cases were created for the **student role**.

Table 5. Student Role Requirements

No.	Role	Feature	Description
1		Login	Access the E-Sorogan system
2		Quran Verse Pop-Up	Display pop-up with Quran verses

3	Student	KRS (Study Plan Card)	List of Study Plan Cards
4		Class Schedule	List of class schedules
5		Assignments	List of assignments
6		Forum	List of forums
7		Payment Notifications	Notifications about payments
8		Spada Indonesia	Information page about Spada Indonesia
9		Latest News	Display of latest news
10		Virtual Reality	List of virtual reality resources
11		Announcements	List of announcements
12		Lecture Tutorials	List of lecture tutorial videos
13		Profile	Student profile page
14.		Logout	Logout from system E-sorogan

Example Test Case

Table 6. Test Case Esorogan – 00 – 0A

Module	Login	
Test ID	Esorogan - 00 - 0A	
Test Case Name	Student Login	
Description	Student logs into their account to gain full access to the system.	
Type	Positive	
Scenario	1. Access the system link: https://esorogan.unusa.ac.id/ 2. Click Login 3. Enter NIP / NIM / Email and password 4. Click Login	
Result		
Expected Result	Actual Result	Conclusion
Student successfully logs into the system.		

Table 7. Test Case Esorogan – 00 – 0B

Module	Login
Test ID	Esorogan - 00 - 0B
Test Case Name	Student Login with Incorrect Credentials
Description	Student attempts to log in using an incorrect NIM and password.
Type	Negative
Scenario	1. Access the system link: https://esorogan.unusa.ac.id/

E-ISSN:*2621-4474, ISSN:2621-4458.*

	2. Click Login 3. Enter incorrect NIP / NIM / Email and password 4. Click Login	
Result		
Expected Result	Actual Result	Conclusion
Student fails to log in and a pop-up notification of failed login appears.		

D. Test Case Execution

In this stage, test cases were executed systematically. The test case execution process involves systematic testing, where each test case is executed, the steps followed are recorded, and the results are documented. The execution was performed using four automated testing tools: Selenium, Katalon, UI.Vision, and BugBug. The following tables summarize the results of executing test cases on these tools.

Table 8. Test Case Execution on Selenium

Module	**Virtual Reality**	
Test ID	Esorogan – 04 – 1A	
Test Case Name	Student accesses Virtual Reality menu	
Description	Student opens the Virtual Reality page.	
Type	Positive	
Scenario	1. Access the system link: https://esorogan.unusa.ac.id/ 2. Click **Virtual Reality**	
Result		
Expected Result	Actual Result	Conclusion
Student successfully opens the Virtual Reality page	Page opens successfully, but during replay, the cursor must be positioned over the Virtual Reality menu button; otherwise, the test fails.	Fail

Table 9. Test Case Execution on Katalon

Module	**Virtual Reality**
Test ID	Esorogan – 04 – 1A
Test Case Name	Student accesses Virtual Reality menu

Description	Student opens the Virtual Reality page.	
Type	Positive	
Scenario	1. Access the system link, link: https://esorogan.unusa.ac.id/ 2. Click **Login** 3. Enter NIP / NIM / Email and password 4. Click **Login** 5. Click **Virtual Reality**	
Result		
Expected Result	Actual Result	Conclusion
Student successfully opens the Virtual Reality page	Student successfully opens the Virtual Reality page.	Pass

Table 10. Test Case Execution on UI.Vision

Module	**Virtual Reality**	
Test ID	Esorogan – 04 – 1A	
Test Case Name	Student accesses Virtual Reality menu	
Description	Student opens the Virtual Reality page.	
Type	Positive	
Scenario	1. Access the system link, link: https://esorogan.unusa.ac.id/ 2. Click **Login** 3. Enter NIP / NIM / Email and password 4. Click **Login** 5. Click **Virtual Reality**	
Result		
Expected Result	Actual Result	Conclusion
Student successfully opens the Virtual Reality page	Student successfully opens the Virtual Reality page	Pass

Table 11. Test Case Execution on BugBug

Module	**Virtual Reality**
Test ID	Esorogan – 04 – 1A
Test Case Name	Student accesses Virtual Reality menu
Description	Student opens the Virtual Reality page.
Type	Positive

E-ISSN:*2621-4474, ISSN:2621-4458.*

Skenario	1. Access the system link, link: https://esorogan.unusa.ac.id/ 2. Click Login 3. Enter NIP / NIM / Email and password 4. Click Login 5. Click Virtual Reality	
Result		
Expected Result	Actual Result	Conclusion
Student successfully opens the Virtual Reality page.	Student successfully opens the Virtual Reality page.	Pass

E. Analysis of Test Case Execution Results

At this stage, the test execution results for each tool were analyzed. The previous step involved executing the test cases using Selenium, Katalon, UI.Vision, and BugBug, all of which are automated testing tools employing the capture-and-replay technique. These tools are designed to automate user interface (GUI) testing for websites.

For the Student role, 21 test cases were designed and executed. The results are as follows:

- Selenium**:** Out of 21 test cases, 16 passed and 5 failed. The failed test cases (Esorogan – 02 – 1A, Esorogan – 02 – 2A, Esorogan – 02 – 3A, Esorogan – 02 – 4A, and Esorogan – 04 – 1A) were related to course modules and Virtual Reality, including KRS, class schedule, assignments, forum, and Virtual Reality menus. Failures occurred because, during replay, the system did not display the expected page if the cursor was not positioned correctly on hover menus.
- Katalon**:** All 21 test cases passed, indicating that most tested features functioned as expected.
- UI.Vision**:** All 21 test cases passed, showing consistent performance across the tested features.
- BugBug**:** All 21 test cases passed, confirming that the system's features functioned correctly.

Overall, the GUI quality of the Esorogan system at UNUSA is satisfactory**,** and all system features operate as intended. Some tools, however, faced limitations in recording hover and drop-down menu interactions.

F. Tools Performance Analysis

At this stage, the aim is to evaluate the performance of the testing tools used during the execution of the test cases. Two comparative tables are provided: Test Constraints and Execution Time. The Test Constraints Table is intended to compare the obstacles encountered during the execution of test cases for each tool. The table is as follows:

Test Constraint	Test ID	Automated Tools			
		Selenium	Katalon	UI.Vision	BugBug

Sensitivity to hover/drop-down features	Esorogan – 02 – 1A, Esorogan – 02 – 2A, Esorogan – 02 – 3A, Esorogan – 02 – 4A, dan Esorogan – 04 – 1A.	1	0	0	0

Based on the Test Constraints Table above, the results indicate that the main challenge during test case execution occurred with Selenium, which is sensitive to hover and drop-down menu features. This caused five test cases to fail when the cursor was not positioned correctly during replay. In contrast, Katalon, UI.Vision, and BugBug did not encounter this issue, indicating better handling of hover interactions in these tools.

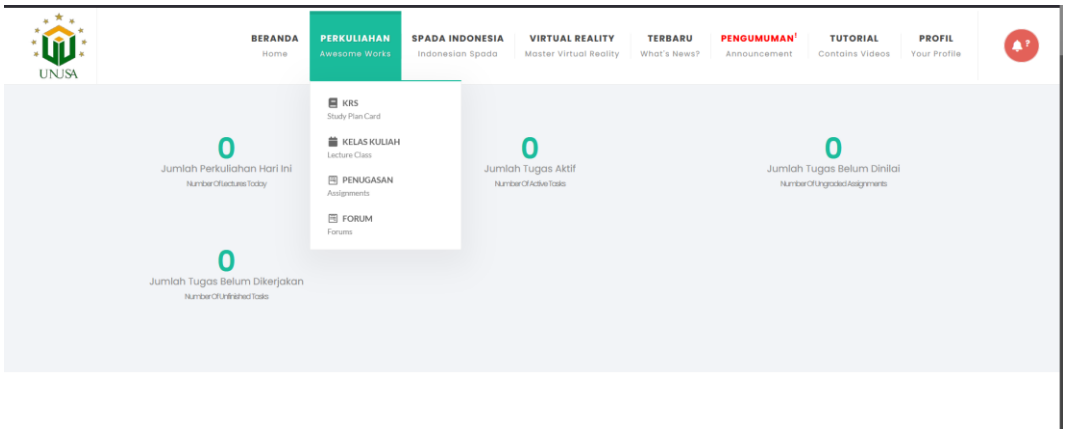

Figure 2. Failure to execute the hover action on the course module drop-down menu in Selenium

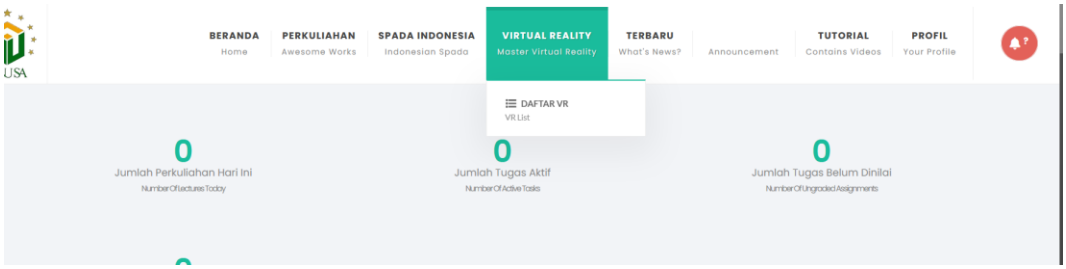

Figure 3. Failure to execute the hover action on the Virtual Reality drop-down menu in Selenium

Selenium has limitations in executing hover actions on drop-down elements, which can result in failure to display the expected results. During replay, if the cursor is manually positioned over the hover feature, the

E-ISSN:*2621-4474, ISSN:2621-4458.*

action succeeds. Other tools such as Katalon, UI.Vision, and BugBug did not experience issues during test case execution.

Additionally, we compared the execution speed of test cases across the four tools. The results are summarized in the Test Execution Time Table below:

Table 12. Test Execution Time

Test ID	Execution Time (Tools) in Seconds			
	Selenium	Katalon	UI.Vision	BugBug
Esorogan – 00 – 0A	9 s	4 s	19 s	18 s
Esorogan – 00 – 0B	4 s	2 s	14 s	16 s
Esorogan – 00 – 1A	7 s	4 s	17 s	19 s
Esorogan – 00 – 2A	7 s	5 s	18 s	21 s
Esorogan – 00 – 3A	6 s	6 s	23 s	23 s
Esorogan – 01 – 1A	8 s	7 s	18 s	19 s
Esorogan – 01 – 2A	8 s	7 s	21 s	20 s
Esorogan – 01 – 3A	9 s	7 s	21 s	20 s
Esorogan – 01 – 4A	9 s	7 s	21 s	20 s
Esorogan – 01 – 5A	8 s	6 s	23 s	26 s
Esorogan – 02 – 1A	-	6 s	17 s	22 s
Esorogan – 02 – 2A	-	8 s	21 s	20 s
Esorogan – 02 – 3A	-	7 s	21 s	24 s
Esorogan – 02 – 4A	-	6 s	20 s	20 s
Esorogan – 03 – 1A	8 s	6 s	22 s	27 s
Esorogan – 04 – 1A	-	8 s	21 s	29 s
Esorogan – 05 – 1A	7 s	5 s	20 s	20 s
Esorogan – 06 – 1A	9 s	5 s	23 s	21 s
Esorogan – 07 – 1A	10 s	7 s	21 s	30 s
Esorogan – 08 – 1A	6 s	5 s	20 s	22 s
Esorogan – 09 – 1A	6 s	5 s	16 s	19 s
Total	121	123	417	456

G. Comparison of Test Execution Time

The comparison of test execution times was conducted using four automated testing tools across 20 test cases. The average execution time for each tool was calculated using the formula:

Average Test Execution Time

Average Test Execution Time: $\frac{Total\ Execution\ Time}{Number\ of\ Test\ Case}$ ……..(1)

The results are as follows:

- Selenium: (9+4+7+7+6+8+8+9+9+8+8+7+9+10+6+6)/20 = 121/20 = 6.05 seconds
- Katalon: (4+2+4+5+6+7+7+7+7+6+6+8+7+6+6+8+5+5+7+5+5)/20 = 123/20 = 6.15 seconds
- UI.Vision: (19+14+17+18+23+18+21+21+21+23+17+21+21+20+22+21+20+23+21+20+16)/20 = 417/20 = 20.85 seconds
- BugBug: (18+16+19+21+23+19+20+20+20+26+22+20+24+20+27+29+20+21+30+22+19)/20 = 456/20 = 22.8 seconds

Although Selenium has the lowest average time (6.05 s), it had 5 test cases that failed due to its inability to execute hover actions reliably. Therefore, the fastest and most reliable average test execution time was achieved by **K**atalon at 6.15 seconds. UI.Vision and BugBug recorded significantly longer execution times, indicating lower efficiency compared to Selenium and Katalon.

H. Tools Performance Evaluation

The performance of each automated testing tool was evaluated based on reliability and execution speed:

1. Selenium:
    - Showed generally consistent performance with most test cases passing.
    - Experienced failures on 5 test cases due to limitations in executing hover drop-down actions, indicating potential issues in more complex UI testing scenarios.
2. Katalon:
    - Achieved excellent performance with an average execution time of 6.15 seconds**.**
    - All test cases passed successfully, demonstrating both efficiency and reliability in executing automated test cases.
    - Proven to handle complex UI interactions effectively, making it the preferred tool in this comparison.
3. UI.Vision and BugBug:
    - Successfully executed all test cases without functional failures.
    - However, both tools recorded significantly longer execution times (20.85 s and 22.8 s, respectively), indicating lower efficiency**.**
    - Better suited for scenarios where execution time is not critical, and reliability takes priority.

V. CONCLUSIONS

Based on the research conducted, it can be concluded that a total of 21 test cases were designed and executed. Using Selenium, 16 out of 21 test cases passed while 5 failed, particularly in the modules related to courses and virtual reality, including KRS, class schedules, assignments, forums, and virtual reality menus. In contrast, Katalon Studio, Ui.Vision, and BugBug successfully executed all test cases, indicating that most features in the Esorogan Unusa system function as expected. Overall, the system's GUI quality

is adequate, although Selenium showed limitations in handling hover drop-down actions, which caused some test failures. In terms of performance, Katalon Studio demonstrated the fastest average execution time at 6.15 seconds, followed by Selenium, which, despite having a slightly lower average, was affected by several failed test cases. Ui.Vision and BugBug executed all tests without issues but required longer average times of 20.85 and 22.8 seconds, respectively, suggesting lower efficiency compared to Katalon and Selenium. Therefore, Katalon is considered the most efficient and reliable tool, followed by Selenium, while Ui.Vision and BugBug are more suitable for scenarios where execution time is not critical. For future research, it is recommended to conduct testing in different environments, such as desktop or mobile applications, and to analyze licensing costs and potential long-term time efficiency for each tool.

VI. REFERENCES

[1] H. V. Gamido and M. V. Gamido, "Comparative review of the features of automated software testing tools," *Int. J. Electr. Comput. Eng.*, vol. 9, no. 5, pp. 4473–4478, 2019, doi: 10.11591/ijece.v9i5.pp4473-4478.

[2] D. Setiawan, M. A. Fadhillah, A. Wibawa, I. Sugiarto, A. Mulyana, and I. Kusyadi, "Pengujian Black Box pada Aplikasi Perpustakaan Berbasis Web Menggunakan Teknik Equivalence Partitioning," *J. Teknol. Sist. Inf. dan Apl.*, vol. 3, no. 2, p. 95, 2020, doi: 10.32493/jtsi.v3i2.3955.

[3] M. T. Abdillah *et al.*, "Implementasi Black box Testing dan Usability Testing pada Website Sekolah MI Miftahul Ulum Warugunung Surabaya," *J. Ilmu Komput. dan Desain Komun. Vis.*, vol. 8, no. 1, 2023.

[4] M. Swillus and A. Zaidman, "Sentiment overflow in the testing stack: Analyzing software testing posts on Stack Overflow," *J. Syst. Softw.*, vol. 205, p. 111804, 2023, doi: 10.1016/j.jss.2023.111804.

[5] H. Rusli, "Analisa perbandingan black-box automated testing dan manual testing pada aplikasi accmart," 2020.

[6] A. Arcuri, "Automated Black- And White-Box Testing of RESTful APIs with EvoMaster," *IEEE Softw.*, vol. 38, no. 3, pp. 72–78, 2021, doi: 10.1109/MS.2020.3013820.

[7] N. U. Hasanah, *Kajian otomatisasi pengujian gui: selenium ide, Uipath Studio, Katalon studio*. 2022. [Online]. Available: https://repository.uinjkt.ac.id/dspace/handle/123456789/66772%0Ahttps://repository.uinjkt.ac.id/dspace/bitstream/123456789/66772/1/NURUL USWATUN HASANAH-FST.pdf

[8] M. Cernat, A. N. Staicu, and A. Stefanescu, "Improving UI test automation using robotic process automation," *ICSOFT 2020 - Proc. 15th Int. Conf. Softw. Technol.*, no. Icsoft, pp. 260–272, 2020, doi: 10.5220/0009911202600267.

[9] M. M. Muhtadi, M. D. Friyadi, and A. Rahmani, "Analisis GUI Testing pada Aplikasi E-Commerce menggunakan Katalon," *Pros. Ind. Res. Work. Natl. Semin.*, vol. 10, no. 1, pp. 1387–1393, 2019.

[10] S. Melia and F. P. Putra, "Comparative Analysis of Automated Testing Tools on GUI WEB-Based Applications Analisis Perbandingan Tools Pengujian Otomatis pada GUI Aplikasi Berbasis WEB," pp. 267–273, 2023.

[11] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, *Approaches and Tools for Automated End-to-End Web Testing*, 1st ed., vol. 101. Elsevier Inc., 2016. doi: 10.1016/bs.adcom.2015.11.007.

[12] H. Anjum *et al.*, "Otomatisasi Pengujian Aplikasi Web Toko Sembako Menggunakan Selenium IDE," *Log. J. Ilmu Komput. dan Pendidik.*, vol. 1, no. 2, pp. 303–309, 2023, [Online]. Available: https://journal.mediapublikasi.id/index.php/logic/article/view/1654

[13] R. P. Octavially, R. R. Riskiana, K. A. Laksitowening, D. S. Kusumo, M. Adrian, and N. Selviandro, "Test Case Analysis with Keyword-Driven Testing Approach on Angkasa Website Using Katalon Studio Tools," *Ultim. J. Tek. Inform.*, vol. 13, no. 2, pp. 134–141, 2022, doi: 10.31937/ti.v13i2.2391.

[14] E. Pelivani and B. Cico, "A comparative study of automation testing tools for web applications," *2021 10th Mediterr. Conf. Embed. Comput. MECO 2021*, pp. 7–10, 2021, doi: 10.1109/MECO52532.2021.9460242.

[15] Meenu and Y. Kumar, "Comparative Study of Automated Testing Tools: Selenium , SoapUI, HP Unified Functional Testing and Test Complete," *J. Emerg. Technol. Innov. Res.*, vol. 2, no. 9, pp. 42–48, 2015, [Online]. Available: www.jetir.org

[16] S. S. Rahayu *et al.*, "Analisis Penggunaan Tools Automation Testing pada Aplikasi : Systematic Literature Review," vol. 8, pp. 106–116, 2024.

[17] B. Tech, "OVERVIEW AND ANALYSIS OF AUTOMATED TESTING TOOLS: RANOREX, TEST COMPLETE, SELENIUM DHEERAJ KAKARAPARTHY IRJET Journal OVERVIEW AND ANALYSIS OF AUTOMATED TESTING TOOLS: RANOREX, TEST COMPLETE, SELENIUM DHEERAJ

KAKARAPARTHY," *Int. Res. J. Eng. Technol.*, pp. 1575–1579, 2017, [Online]. Available: www.irjet.net
[18] A. C. Barus and S. Leo, "Android Comparative Study of Automated Testing Tools for Android," *JTIIK J. Teknol. Inf. dan Ilmu Komput.*, vol. 6, no. 6, pp. 645–654, 2019, doi: 10.25126/jtiik.20196953.
[19] J. Wang, Y. Wang, and S. Cheng, "A survey on GUI testing techniques for mobile applications," IEEE Access, vol. 9, pp. 4533–4548, 2021, doi: 10.1109/ACCESS.2021.3049440.
[20] A. M. Memon, "Automatically repairing event sequence-based GUI test suites for regression testing," ACM Trans. Softw. Eng. Methodol., vol. 29, no. 4, pp. 1–35, 2020, doi: 10.1145/3380943.
[21] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Capture-replay tools for mobile and web testing: A systematic review," Softw. Test. Verif. Reliab., vol. 32, no. 1, pp. 1–34, 2022, doi: 10.1002/stvr.1763.
[22] Y. Li, C. Xu, and L. Zhang, "DeepTest: Automated testing for deep-learning-based autonomous systems," Proc. IEEE Int. Conf. Softw. Eng., pp. 101–112, 2021, doi: 10.1109/ICSE43902.2021.00020.
[23] A. Anand et al., "Efficient GUI testing using model-based and machine-learning approaches," IEEE Trans. Softw. Eng., vol. 48, no. 3, pp. 789–803, 2022, doi: 10.1109/TSE.2020.3048802.
[24] L. K. Sharif, S. S. Idrus, and A. A. Rahman, "Evaluating Selenium WebDriver performance using multiple browsers," Indones. J. Electr. Eng. Comput. Sci., vol. 24, no. 2, pp. 998–1007, 2021, doi: 10.11591/ijeecs.v24.i2.pp998-1007.
[25] M. Gutiérrez et al., "Test automation for web applications: A review of tools, techniques, and challenges," J. Syst. Softw., vol. 190, p. 111327, 2022, doi: 10.1016/j.jss.2022.111327.
[26] P. N. Robles, F. P. Miller, and R. L. Smith, "Assessing maintainability problems in GUI test scripts," IEEE Softw., vol. 37, no. 4, pp. 62–71, 2020, doi: 10.1109/MS.2020.2974979.
[27] A. O. Olanrewaju and J. A. Adu, "Comparison of web test automation frameworks: Selenium, Cypress, and Playwright," IEEE Africon, pp. 812–817, 2023, doi: 10.1109/AFRICON55921.2023.10247652.
[28] C. Pinto, R. Dias, and A. Sampaio, "Model-driven automation for GUI testing in web applications," IEEE Int. Symp. Softw. Reliab. Eng., pp. 55–66, 2021, doi: 10.1109/ISSRE52982.2021.00018