Addis Ababa University

College of Natural Science

Web Element Locator Algorithm for Dynamic Web Application Testing Using Machine Learning

Mikiyas Bayew Bekele

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

August, 2021

Addis Ababa University

College of Natural and Computational Sciences

Mikiyas Bayew Bekele

Advisor: Mesfin Kifle (PhD)

This is to certify that the thesis prepared by Mikiyas Bayew, titled: *Web Element Locator Algorithm for Dynamic Web Application Testing Using Machine Learning* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name		Signature	Date
Advisor:	Mesfin Kifle (PhD)	_____	_____
Examiner:	Solomon Atnafu (PhD)	_____	_____
Examiner:	Minale Ashagrie (PhD)	_____	_____

Abstract

Software testing is one of the software development life cycle for detect or discovers errors of the software. It ensures the correctness, completeness, and quality of software that has been developed. Automated software testing is a good strategy for reducing software testing effort during web application testing. During web applications, testers develop a web element locator to find web elements on a page using a query through test scripts. Developers apply changes to their web applications to meet new requirements, adding new functionalities, fixing bugs, *etc.* During those times web elements attribute like identifier (ID), name and classes of the web application dynamic and keeps changing. A simple modification of the application programming interface (API) affects locators, which leads to unable to select the desired web elements on the web application that may cause a test to fail. The main reason for the appearance of test case breakage is the failure of the element's locator on the dynamic web application (DWA). It disturbs those who use test automation so much. To repair these fragility problems, test engineers must debug and rewrite those test cases. Because the locators that are used to select the web element may no longer be valid in the updated version. This process takes additional time for testing dynamic web applications.

In this research, to improve the accuracy and performance of DWA testing, we proposed a web element locator algorithm (WELA) using machine learning which automatically identifies web elements. The algorithm covers structural, logical, and presentation types of changes of web elements that may cause the breakage of the web element locator. It can identify a similar pattern based on web element features to adjust the locator according to the change. This makes the test more reliable and maintainable, by reducing the time and effort required to maintain web element locators. The testing process begins after the correct web elements have been identified.

An experiment is performed in ten web applications to prove the effectiveness of WELA in terms of accuracy and performance. The result is promising, which shows the proposed approach effectively repairs 97% of broken web test scripts and generates the test with the relatively shortest execution time on the evolved versions of a DWA.

Keywords: Automated Testing, Dynamic Web Application Testing, Web Application Change, Web Element Locator, Machine Learning

Acknowledgment

First and foremost, I would like to thanks almighty God and His Holy Mother St. Mary, the Seven Arc Angles, and Priests in Heaven, for his remedies for my sickness and his responses to my prayer in every bit of my troubles throughout my life and accomplishment of this thesis.

I would like to express my deep gratitude to my advisor Dr. Mesfin Kifle, for his guidance, enthusiastic encouragement, and convenient critiques of this research in keeping my progress on schedule even in the pandemic period. Whenever I was lost, he guided me in the right direction and pushed me to do what I needed to do.

I would also like to thank my family for their continuous love and respect, support, and encouragement to make my dream become success. Their support and care helped to overcome setbacks and stay focused on my graduate study.

Last but not least, I would like to express my gratitude to all my friends, with whom I have shared moments of deep anxiety as well as great joy throughout this research work. A kind word for my friends that always managed to make me feel special. Thanks, all my friends, for always being there with me. I express my special thanks to Duyen Phuc Nguyen, who assisted me in providing machine learning data and valuable technical support for this research.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acronyms and Abbreviations

- AI Artificial Intelligence
- AJAX Asynchronous JavaScript and XML
- API Application Programming Interface
- ASP Active Server Page
- ATA-QV Automatic Test and Analysis qui vicino (qui vicino an Italian term that translates to “hear out” or “near here”)
- AUT Application Under Test
- CD Change Detection
- CSS Cascading Style Sheets
- CSV Comma Separated Values
- COLOR Correct Locator Recommender
- DOM Document Object Model
- DT Decision Tree
- DWA Dynamic Web Application
- GUI Graphic User Interface
- HTML Hypertext Markup Language
- HTTP Hypertext Transfer Protocol
- ID Identifier
- IDE Integrated Development Environment
- IEEE Institute of Electrical and Electronics Engineers
- KNN K-Nearest Neighbor
- LOC Line of Code
- ML Machine Learning
- PHP Pre-Processor Hyper Text
- PPMA PHP Password Manager
- QTP Quick Test Professional
- ROBULA Robust Locator Algorithm
- SDLC Software Development Life Cycle
- SQL Structured Query Language
- SUT System Under Test

- SVM Support Vector Machine
- UI User Interface
- URL Uniform Resource Locator
- WATER Web Application Test Repair
- WELA Web Element Locator Algorithm
- XML Extensible Markup Language
- XPath XML Path Language

1. Introduction

1.1 Background

One of the most important steps in the software development life cycle (SDLC) is software testing. It is a method for identifying the correctness, completeness, and quality of software that has been developed [1]. Web application testing is a software testing practice to test web applications for potential bugs. It includes a set of activities conducted with the intent of finding errors in software to be corrected before the product is released to end users [2]. Because it gives a confident quality of the final product, confirms that the application has no errors in the code, verifies how the user can work with the application and ensures that the end product is easy to use.

Web application testing faces a problem in manual testing when they need to create test cases and run test cases repeatedly especially if the application versions change frequently [3]. To overcome those problems automated testing comes up as a solution that focuses on replacing manual human activities with systems or devices that enhance the efficiency of the software testing process. It uses the assistance of tools, scripts, and software to perform test cases by repeating pre-defined actions [4]. Automated testing develops a web element locator that finds and returns web elements on a page using a given query through test scripts. As the dynamic nature of the web application increases, web element locator like ID, extensible markup path language (XPath), and name of the page will be different each time the page is visited [5], when it is visited by a different user or for successive release of a web application. The modification of the API has an impact on locators that leads to unable to select the desired elements on the web page [6].

It is important to have good locators to find the correct web elements, making sure that tests are faster, more accurate, and more maintainable. Often, we end up working with element attribute not found, index-based locator not found, or inconsistent web graphic user interface (GUI) elements, such as those lacking unique ID or class names, automatically generated ID not found [7]. Those web element attributes vary from release to release, making it hard or even impossible to find the correct elements on the page [8]. It also leads to the automated script has to find these web elements take up a lot of time [5].

Failure to identify web elements may result in less reliable tests or test code fragility problems [9].To make that test are faster and more accurate, we propose a web element locator algorithm (WELA) for DWA testing using machine learning (ML) techniques which automatically identifies web elements. Automatic recognition of these elements plays an important role in tasks such as automated web applications testing [10].

ML as a subdomain of artificial intelligence (AI) which is widely used in various stages of the SDLC [11], especially for automated software testing processes. It helps systems to learn and apply the learned knowledge in the future which helps software testers with more accurate knowledge. The DWA would be tested and it is trained to a system from a human perspective in such a way that the algorithm is able to understand which web element locator to be followed for every element on the web application so that next time the test will be based on those input elements. The algorithm learns what actions to be done for a particular web element and can detect or predict the change to adapt those changes for the locators in which the web element changes.

1.2 Motivation

Nowadays, developing an "error-free" software product becomes a key concern for software developers and research communities. Software testing is one of the most challenging tasks of the software development lifecycle [12]. Many organizations and businesses tend to spend a whole or most of the efforts and resources on other SDLC. On the other hand, they spent a short time testing the software product [13], that affects the quality of the software product. So, developers need effective automated testing to ensure that their software works well, everywhere within a minimum effort. Creating an effective web element locator is a significant challenge for successful automated DWA testing. Thus, issues motivated us to work and contribute to the automated testing especially, for locating web element for web application testing.

Software systems have become a vital aspect of the economy in multiple fields, including education, banking, healthcare, communication, and more. However, whenever they fail, there are economic consequences, customer dissatisfaction, the loss of financial assets, and even the loss of human lives. According to a study conducted by a software testing business, software failure affected 3.6 billion individuals in 2017 and resulted in $1.7 trillion in financial

losses [14]. A good testing practice would have eliminated these issues because it ensures that bugs and issues are detected early in the life cycle of software product. These issues inspire us to investigate more on the testing particularly in dynamic web testing.

1.3 Statement of the Problem

Web element locator is the core process of web test automation. For handling a web element, we first need to identify the web element correctly. It is like the way we recognize someone who has some varied unique characteristics or identifications. Identifying web elements has always been required an accurate and efficient approach for web testing [15]. Locators are specific commands used by test automation tools to identify the web elements on the GUI before performing actions on them. A simple modification of the API test has an impact on locators that leads to unable to select the desired web elements on the web application may cause a test fail [6, 16]. Furthermore, many test scripts are manually handled which increases the cost and time for testing [5].

The rapid development of web applications causes some broken taste cases. The main reason for the appearance of test case breakage is the failure of the element's locator on the web application like non-selection, mis-selection, form data problems, and obsolete content problems [7, 17]. The fragility of GUI tests has been a big problem that disturbs those who use test automation [18]. To repair these fragility problems in automated testing, test engineers must debug and rewrite those test cases. This process takes additional time and leads to slow down the time for testing a given software. Thereby, we can assert that the more efficient the locator, the more stable will be the automation script.

Automated web testing tools like selenium, supports several techniques for identifying web elements including names, ID, absolute XPath, relative XPath, and more [19]. However, when developers update the software, they often need to modify the test scripts accordingly [20]. A robust XPath locators' algorithm is also used for automated web application testing [21]. However, an XPath expression for an element would frequently contribute to longer running tests, and handling dynamic objects or web elements were one of the most significant problems in Selenium, and Using XPath expressions tends to take much longer to run [22, 23]

In general, the testing stability of web application, are often impacted by elements within them that are either dynamic by the developers or more commonly, changes are made directly to

the application like functionality addition, functionality modification, functionality deletion. Non-ML test scripts cannot automatically adapt and overcome those changes. This inability results in test failures, brittle tests, and inconsistent test data [24]. More specifically the following research questions need to be addressed.

- What are the problems of the web element locator for DWA testing?
- Witch ML algorithm can cope with DWA issues especially for the identification of web elements?
- What type of features are more appropriate for ML model training?

1.4 Objectives

General Objective

The general objective of the research is to develop a web element locator algorithm for DWA testing using ML.

Specific objectives

To realize the general objective of the research, the following specific objectives are identified.

- Review Literature concerning web element, DWA testing, web element locators, and ML.
- Identify the significance features to train ML model for locating web elements.
- Design web element locator algorithm for DWA testing.
- Implement and evaluate the performance and accuracy of the algorithm through a simulation.

1.5 Methodology

To achieve the general and specific objectives mentioned above, a design science research approach will be used. The foundations for the application of design science research as a research method and the methods formalized by several authors are presented. From the different process models and descriptions with diagrams of the design science research processes presented by different authors we will be adopted Peffers *et al.* [25]. The reseaon behind is that by considering the method used should address the research question, the

method must be recognized the scientific community, and the method should demonstrate the procedures that were adopted for the research.

As shown in Figure 1.1, the starting point of the research that employs the method is a literature review step. It is to search for existing solutions for web element locators and identify well-established theories that can serve as a basis for the research and wish to propose a new solution to fill the gaps in the existing knowledge. After investigating the gaps, in the design and development step the artifact that will help to find the answer to “how to solve the problem on the identification of web elements with dynamic content?”. Once the algorithm is developed at the evaluation step the algorithm will be tested, not only the algorithm works but also whether its improved performance and accuracy of locating the write web element for DWA will be evaluated.

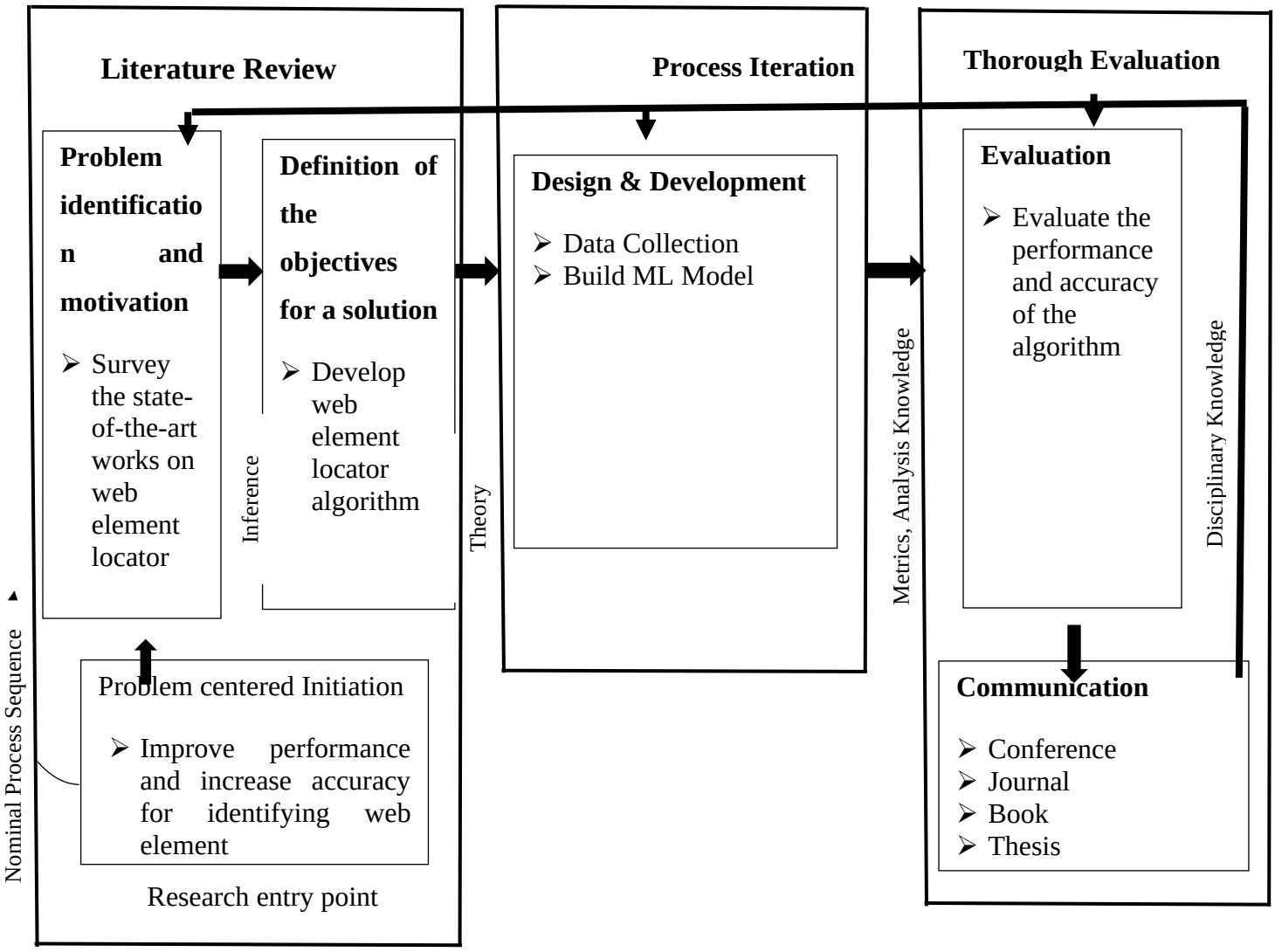

Figure 1.1: Design Science Research Process for Web Element Locator

Data Collection

The first step is to find training data to build an effective ML model for a given DWA testing problem. We collected 5,280 web elements from 250 DWA using the scraping tool beautiful soup from SourceForge and Alexa database. The criteria to select these web applications have been their number of web elements, the number of attributes they have, web elements that have changed their attribute values, and their structural complexity. The complexity of the DOM structure defines how far web elements they settled down in the structure.

1.6 Scope and Limitations

This research work aimed to design a web element locator algorithm. The proposed algorithm particularly focuses sample of ten open-source web applications with two successive releases from SourceForge, as well as a common example of dynamic applications such as Amazon, Gmail, and Google Search.

1.7 Application of Results

The result of this research work will contribute to ongoing researches in the automated testing area for DWA. Because it is important to have good locators to find the correct web elements, making sure that tests are faster, more accurate, and more maintainable. The main benefits of the algorithm in test automation are finding bugs on the early stages of development, and reusing test code across successive releases of the application under test (AUT).

Currently, large information technology companies such as Google, Amazon and web application developers use different automation tool like Selenium [19]. But they are likely to spend a big portion of their test development time trying to locate web elements on a web application. So, this work will save time and effort in maintaining test scripts for web application testing**.** Another important thing is to reduce dependency on subject matter experts by automatically selecting the test to execute according to the scenario and dynamically refining the task scope according to changes in the test strategy.

1.8 Organization of the Rest of the Thesis

The rest of this thesis is organized as follows. This section givers an overview of the contents of each Chapter. Chapter Two presents the literature review of important concepts related to web element locator for DWA testing. Chapter Three presents recent research works that will be investigated to understand web element locator is explored so far and what are the limitation and gaps that need to be addressed. Chapter Four explains a detailed description of the architectural design of the proposed algorithm, components, subcomponents. Chapter Five presents topics that are concerned with describing the implementation details of the proposed algorithm and evaluation. Finally, Chapter Six summarizes the thesis with the major concept raised in this research work, the research contributions, conclusions, and potential future works.

2. Literature Review

This Chapter deals with the review of important concepts related to web element locator, aiming to guide the proposed work understandable by readers and practitioners in the field of DWA testing. It discusses basic concepts about the web application, automated testing, web application testing techniques, and the way for locating web elements for web application testing. Generally, it is a critical summary of the background knowledge on the topics that are the basis for this thesis research work.

2.1 Web Application

A web application is a program stored on a remote server and it can be accessed using a web browser interface over the Internet using hypertext transfer protocol (HTTP) [26]. It typically builds the front end with HTML, CSS, and JavaScript, which are supported by major browsers. Stephane *et al.* [5] define that web applications are computer programs that allow website visitors to submit and get data from a database using their favorite web browser over the Internet. It is primarily intended to execute specific activities and achieve specified goals and objectives. Depending on the requirements and aims, these features can be basic or complicated.

A web application is a collection of static and DWA [27]. A static web application does not change when a site visitor requests it. The web server sends the page to the requesting web browser without modifying it. A DWA, on the other hand, is updated by the server before being sent to the requesting browser. The changing nature of the page is why it's called dynamic.

2.1.1 Static Web Application

A static web application comprises a set of related HTML pages and files hosted on a computer running a web server. A web server is a software that serves web applications in response to requests from web browsers. It generated a page request when a visitor clicks a link on a web application, selects a bookmark in a browser, or enters a uniform resource locator (URL) in a text box. The page designer determines the final content of a static web application, which does not change when the page is accessed. When a web server receives a request for a static web application, the server sends the application directly to the requesting

browser. When a dynamic application is requested, the webserver responds differently than when a static web application is requested. It sends the page to an application server for processing. The application server reads the code on the page, completes the page according to the code's instructions, and then removes the code. As a result, the application server returns a static page to the web server, which then sends it to the requesting browser [28].

2.1.2 Dynamic Web Application

A DWA generates pages in real-time and sent a response from the server to the client based on the client request [29]. Based on this response the client-side will take the appropriate action. The user changed and updated the contents and layout of a DWA every time. DWA changes their structure and content during the process of viewing that page [30]. Sometimes ID and classes of the web element keep changing. Such web elements are dynamic web elements. These are database-based elements, and they update the values of these elements whenever the database is updated.

According to the IEEE definition, a dynamic web application contains web applications that are generated in real-time. These pages include Web scripting code, such as pre-processor hypertext (PHP), asynchronous JavaScript and XML (AJAX), and active server page (ASP).

Figure 2.1 [31], shows the communication flow how data be access from DWA. A database query is a command to extract data from a database. This query comprises search criteria expressed in a database language called structured query language (SQL). Through the use of a database driver, an application server can communicate with the database [31]. Between the application server and the database, database driver software functions as an interpreter. After the driver establishes communication, the query is executed in the database and a record set is created. A record set is a set of data extracted from one or more tables in a database. They returned the record set to the application server, then the application server inserts data on the page that passes the page to the web server finally web server sends the finished page to the request browser.

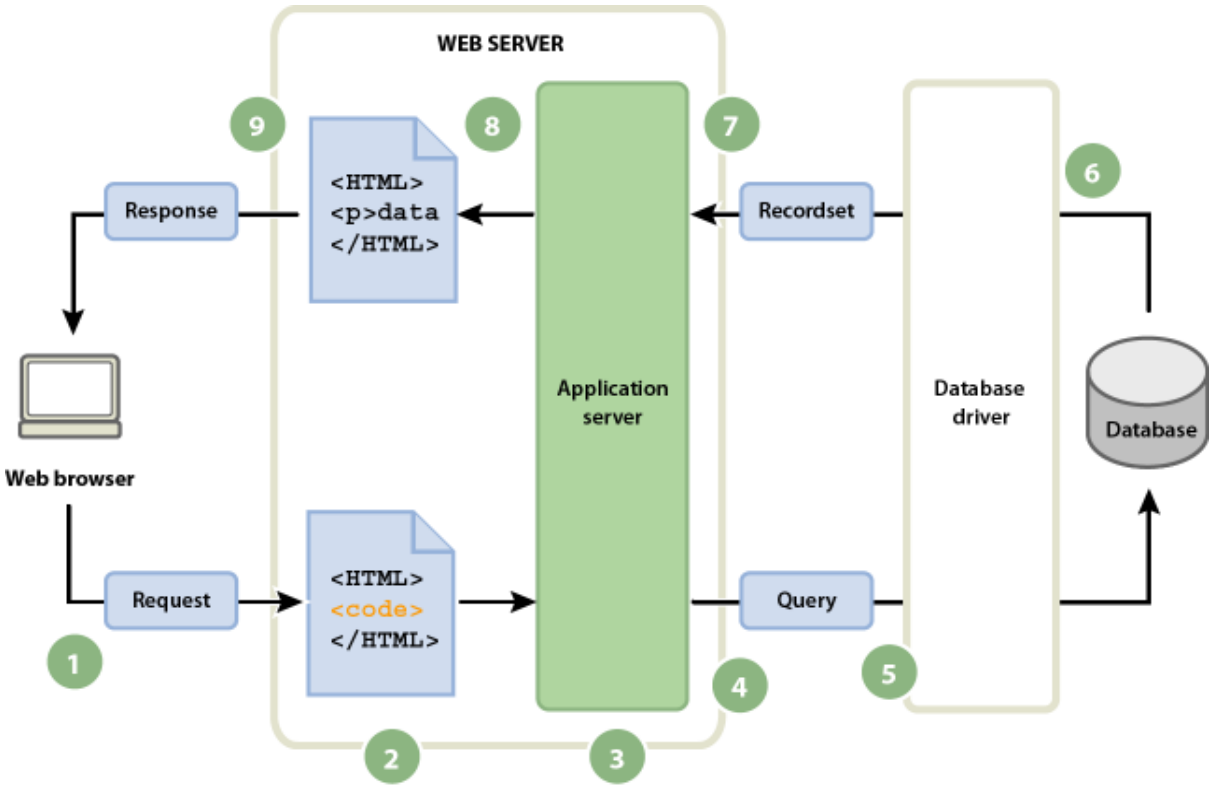

Figure 2.1: Access Database in Dynamic Web Application

Dynamic web application access information from a database. Therefore, to alter the content of this application, the webmaster may only need to update a database record. The cost of developing a DWA is higher at first, but the benefits are many. It can allow the web application owner to edit and add new material to the web application at a basic level. DWA is used when an application requires information changes frequently, such as stock prices or weather forecasts [31].

2.1.3 Web Application Features

Web applications are distinguished from standalone applications by their enormous customer base, heterogeneous execution environment, heterogeneous languages used for component development, heterogeneous operating systems, faster maintenance rate, multi-tier architecture, transactional concurrency, and dynamic state changes [32] as shown in Figure 2.3. Such complexities inherent with web applications make the testing of web applications a challenging job, thus establishing an obvious need for more sophistication in web application testing.

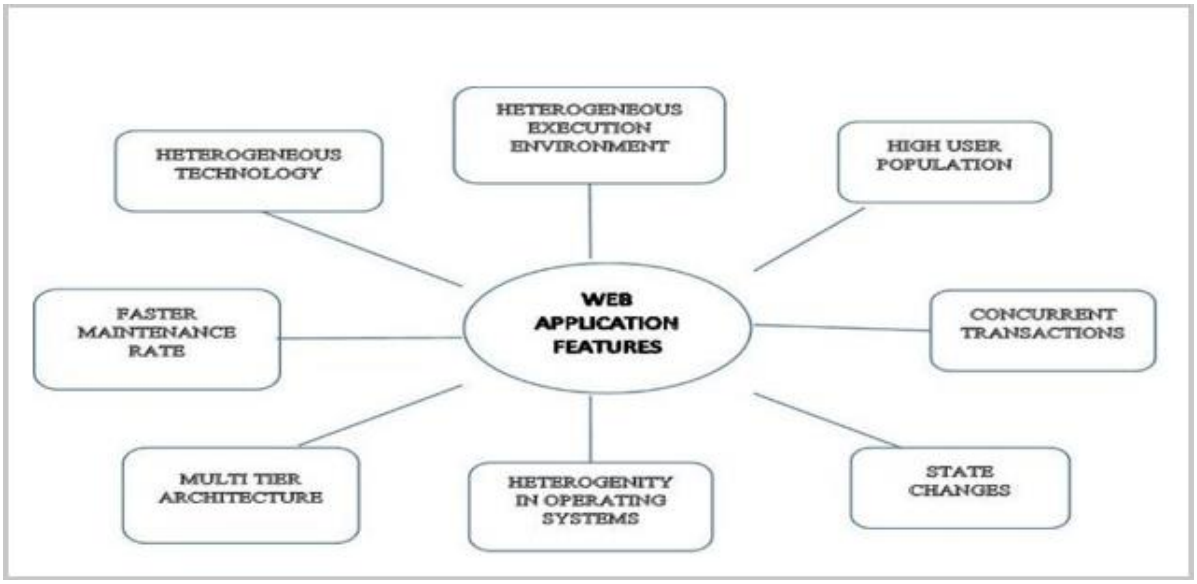

Figure 2.2: Web Application Feature

2.2 Web Application Testing

Web testing is the name given to software testing that focuses on web applications [33]. Web application testing is a mandatory skill for software test engineers these days. It checks bottlenecks and performance leakage in the software or web application to be tested.

Web application testing is a quality assurance activity evaluating the system under test (SUT) by observing its execution to reveal failures [1]. When the SUT's external behavior differs from what we expect of the SUT according to its requirements or some other description of the expected behavior, a failure is discovered [34].

2.2.1 Type of Web Application Testing

Web application testing can be divided into two main categories, manual testing, and automated web application testing [3]. Both categories have their strengths and weaknesses.

A. Manual Testing

Manual testing is a testing process that is carried out manually to find defects without the usage of tools or automation scripting. A test plan document is created by the test lead which describes the detailed and systematic approach to testing a software application and which the tester has to be followed. Typically, the test plan includes a complete understanding of what the ultimate workflow will be. To ensure the completeness of testing, test cases or test scenarios are created [3].

The tester runs the test cases manually to see whether they have any defects. In manual testing, the tester has to pretend as an application user and use all kinds of functions available to find out its right behavior [35]. Manual testing can't be used again, no facility for scripting, and few errors may remain undetected. In addition, manually performed software testing is a time-consuming and burdensome task. Therefore, authors have developed various automated methods and tools to help developers test the quality of a web application.

B. Automated Testing

Automated testing focuses on replacing manual human activities with systems or devices that enhance the efficiency of the software testing process. It uses tools, scripts, and software to perform test cases by repeating pre-defined actions [45]. It is suitable for regression testing, performance testing, load testing, or highly repeatable functional test cases.

The test case is a crucial component of the testing process. A test case outlines the conditions under which SUT detects a failure. When a test case reveals a failure, it is successful. A test case embodies the input values needed to execute the SUT [34]. Therefore, test case inputs vary in nature, ranging from user inputs to method calls with the test-case values as parameters. To evaluate the results of test cases, testers must first know where an element is situated, then conduct an action on the web element and examine the result.

2.2.2 Test Script

Automated testing is one of the more popular and available strategies to reduce testing effort. It uses test scripts that will be used later to execute test cases instead of human [36]. Test script is a set of instructions that are performed on a SUT to verify that the system performs as expected. Each test script command is composed of locator, test action, and test data. The test action specifies the action performed on that specific web elements [20]. A test script is part of a test case, and a test case can have either a single test script or multiple test scripts. The script code will typically do the following one or more times. Identify input elements in the user interface (UI), navigate to the required UI component, identify output element, and verify that output elements display the result, read the result from the output element, assert that output value is equal to the expected value, and write the result of the test to a log [37].

A test automation scripting technique is a set of instructions to define the method of testing to be used by the quality assurance team to get the pre-determined or expected test results. It is important to note that test scripting techniques are not mandatory for test case execution, but they help test teams conduct test case executions with higher accuracy and efficiency. The following are the most often used scripting techniques for test automation by quality assurance teams.

A. Linear Scripting Techniques

The linear scripting approach is a simple record and playback used by a test engineer to automate test case of a system. John Kent [38] explains the idea behind the linear technique, which is simply to set the test tool to the record mode while performing actions on the SUT. The generated recorded script consists of a series of testing instructions using the programming language supported by the tool. Linear scripting technique is easy to use and does not need to have any programming skills. The main disadvantage is the recorded script can only work under the same conditions as when it was recorded the first time. If a simple error happened or unexpected normal events during a test run, it will not be handled correctly by the test script, and it is not reliable enough, even if the application has not changed [39]. One of the major limitations of record and replay tools are if the script author who needed to repeat an action many times would need to record that action and maintain each one of the actions individually.

B. Structural and Shared Scripting Techniques

Structured scripting technique uses structured programming instructions, which either be control structures or calling structures [39]. It uses control structures in the scripts. These control structures enable testers to control the flow of the test script. The control structures are typical ‘if-else’, ‘switch’, ‘for’, ‘while’ statements that help in implementing decision making in the script, to perform some tasks iteratively and capacity to call other common functions that house commonly needed functionality [40]. The most important advantage of the structured technique is that the test script can validate for specific conditions to determine if the executed test passed or failed according to these conditions. However, it requires testing skills as well as programming skills.

C. Data-Driven Scripting Techniques

This technique separates data from scripts and stores it as files on a disk in an external repository. As a result, the test script contains the programmed code. It may need this when the data needs to be varied over the test run.

One benefit of this approach is that it makes it possible to modify a test without actually having to change the test case's code. This makes it possible to separate responsibility for creating tests between test engineers who have coding skills and those who don't. Those with coding skills can create test scripts and encode the test logic in them, and test engineers who don't have coding skills can create and edit the test data that the test scripts use to test the AUT.

D. Keyword-Driven Scripting Techniques

It is an approach in which the control to check and execute operations is maintained with external data files by dividing all test steps by using reusable keyword functions. So, the test data and the operations or sequence of the test are planned in an external data file and an extra library is needed to interpret this data in addition to the conventional script. It is an extension of data-driven testing. A tester can create test cases by using Microsoft excel using a variety of keywords (*e.g.*, click-element, send-keys verify-element, and store text). As the test gets executed and processes the excel file in the driver script one by one against the keywords present in the excel file [41]. The advantage of this scripting technique is to add new test cases and change to test cases requires no code change. The main disadvantage is that webpages change an element identification or dynamism of web applications to become an issue.

E. Page Object Factory Scripting Techniques

It is an object design pattern, where web pages are represented as classes, and the various elements on the page are defined as variables on the class or used to initialize the web elements. All user interactions can then be implemented as methods in the class. The page object model is a design pattern to create an object repository for web UI elements. According to this method, each web page in the application should have its page class. This page class will find the web elements of that web application and also has page methods that conduct operations on those web elements.

2.2.3 Web Application Testing Techniques

The web application testing technique has specific criteria to cover a particular aspect of the program. It defines different test requirements that a test suite should meet. We can generate test requirements from many parts of the software. e.g., specification and implementation. Web application software testing employs some testing techniques, including coverage testing, structural testing, statistical testing, combinatorial interaction testing, penetration testing, search-based software engineering testing, and machine learning techniques [42].

2.2.4 Web Application Test Breakage

Automated test scripts are prone to changes when web applications evolve. It becomes a challenge for the testers to deal with the rapid evolution of the web application under test. Even minor layout changes may lead to breakages in the test scripts. Hammoudi *et al.* [7] develop a detailed taxonomy that discusses the various web test breakages. It broadly classified this taxonomy into five distinct categories of test breakage:

- Breakages related to locators used in tests
- Breakages related to values and actions used in tests
- Breakages related to page reloading
- Breakages related to changes in user session times
- Breakages related to popup boxes

This research focus is on test breakages related to locator-based changes and values and actions used in tests. Locator-based breakage occurs when an attribute of a web element that was used to find the web element in the previous web application changes in the new web application. Web application test breakages associated with value-based changes are when the input value used by the original test is no longer used on the updated web application.

Page reloading activity and user session timeout are the third and fourth categories of test breakages, respectively. Page-reloading activities test breakages occur when the wait time specified in the test script is not enough to load completely and display the content of a web application. User session timeout problem occurs when the web application becomes inactive during testing after n minutes of inactivity [20].

2.3 Web Element

Web element is an individual entity displayed on the web application, as seen by the end-users. Title headings, buttons, input fields, text areas, text boxes, combo boxes, checkboxes, and other web components that a user sees on a web application are examples of web elements. Multiple attributes, such as ID, CSS path, position, size, and value, are assigned to each element [43]. Web element attributes provide a way of attaching additional information about the element that the browser can use. This data can specify how the element is displayed by the browser or identify and categorize the element. Programming languages typically access web elements as nodes in the DOM. The more sophisticated the application, the more the web element becomes dynamic [5]. The dynamic element is that web element attributes such as ID, class, name, value, and so on are changes when the web application change due to different purposes. In addition, when the user reloads the application, it changes [33]. Database-driven or session-driven dynamic elements are available.

2.3.1 Web Element Operations

To access web elements, we need to perform a set of operations starting with browser actions until the operations performed on those web elements [15]. Operations like Launch the browser to work with a web application we first need to open a web browser, navigate to a particular web page, close the current browser, close all browsers opened during execution, maximize browser, and refresh the browser. To perform most operations with web elements, first it needs to locate the UI elements. Table 2.1 shows some web element operation used to in automated testing.

Table 2.1: Web Element Operations

No	Web Element	Operation on the web element
1	Browser	Launch, Navigate, Refresh the browser, Close browser
2	Web Page	Get page title, Get page URL
3	Link	Check link existence, Click link
4	Button	Click, Check enabled status and Display status
5	Image	General image, Image button Image link

6	Text Area	Return or capture the messages from the web page
7	Checkbox	Select the checkbox, Unselect checkbox
8	Radio button	Select the radio button, Check if it displays the radio button
9	Dropdown	Select an item in the list, Get the item in the count

2.3.2 Classification of Web Element Change

There is a continuous evolution of the web application development process, as the perception about it is changing to something more dynamic. Developers usually apply changes to their web applications to meet new requirements, adding new functionalities, fixing bugs, *etc*. Additionally, web applications include some components that are more dynamic, such as ID and web element classes, which change frequently. These are database-based elements whose values get refreshed every time the database gets updated. Leotta *et al.* [9] define types of changes that in web applications: (1) Structural or layout, and (2) Logical changes.

Structural changes refer to change the page layout and structure like re-styling or change a web element's locator attributes. Cohen *et al.* [44] suggest that the number of structural changes increases in the amount of dynamic content with the traffic volume of the web application. The author expects the changes are because of advertisements and dynamically added content, which is much more common on a high-volume web application.

Logical changes refer to all the changes of active web element components. The effects of change are on the functionality of the web application or changing the logic of the web application under test, such as functionality addition, functionality modification, functionality deletion, *etc*. Usually, the impact of structural change is smaller than the logical change.

Another type of change in web elements is presentation changes, which refer to changes in the attribute of DOM objects [45]. Usually, the changes of DOM objects' attributes refer to changes in the rendering or visualization options of a DOM object. This research focus on those three web application changes of DWA testing. Those web elements change breaks most XPath locators used tools like TestComplete or handwritten using frameworks like selenium web driver for finding web elements during web application testing [19].

2.3.3 Web Element Locator

In automated testing, a web element locator is used to identify all the GUI objects to operate upon and eventually to retrieve web page content that is compared against some oracle to decide whether the test case has passed or not [46]. A Web element locator is an object that finds and returns web elements on a page using a query. In short, locators find web elements, and it also software reference points that allow a tester to pull data from and push data to a web page and measure its response [47]. It needed as human users when interacting with web pages visually: We look, scroll, click, and type through a browser. However, test automation interacts with web pages programmatically. It needs a coded way to find and manipulate those web elements.

locators used in a wide range of areas, namely, web testing [17], web data extraction [48], web annotation, web automation, or web augmentation [49]. The main focus is on web application testing, particularly on DWA testing. Web element locators play a vital role in automated web testing. It provides software identifiers to select and validate elements of the code that we are testing. For example, locating and filling in the input areas of a web page, performing certain computations (*e.g.*, clicking on components), and ensuring that the output is correct (showing the results of web application elements).

Locators need to be checked for correctness and possibly updated at every new release of the web application. So, using efficient strategies for web element identification is critical. Web element not found, non-selection, mis-selection, and a form data problem is the most common error we could get once we run scripts in automated web application testing [6, 7]. Automated web applications perform actions on specified items on different web elements, such as locating a button to click on, a text field to fill in, and comparing the output of these actions to a preset intended outcome.

To perform an action on the web element, firstly it must be located. Basically, there are three types of web elements locator approaches based on how test cases localize the web elements to interact with, and what objects that select [19, 21]. DOM-based, visual, and coordinate-based locators.

A. Document Object Model

The Document Object Model (DOM) is an object-oriented tree representation of a web page that represents the hierarchical document structure of the page, and serves as a programming interface for web pages [12]. Second-generation tools use DOM information to locate web elements [21]. Selenium integrated development environment (IDE) and Selenium WebDriver, for example, take this technique and provide several ways to discover web application elements. It defines how documents are accessed and updated, as well as their logical structure. We can access anything in an HTML or XML document using the DOM.

Most web testing tools use attribute-based locators (*e.g.*, ID, name, *etc.)* or structure-based locators (*e.g.*, XPath). A problem with these locators is that they may be changed even by a trivial software update, such as changing a page layout or element name [46] .

i. ID Locator

ID is a unique identification for an element that is usually assigned by the code developer [8]. The ID should be unique on the page. The most widely used method for selecting elements, getElementById is essential to modern web development. With this handy little tool, we can find and work with any element simply by referencing a unique ID attribute. One of the biggest challenges in locating elements by ID is handling web elements with dynamic ID.

```
id="change-password" class="btn" href="#"> Forgotten Password</a>
driver.findElement(By.id("change-password"));//locating using ID
```

ii. Name Locator: Locating elements by name are very similar to locating by ID, except that we use the **"name="** prefix instead.

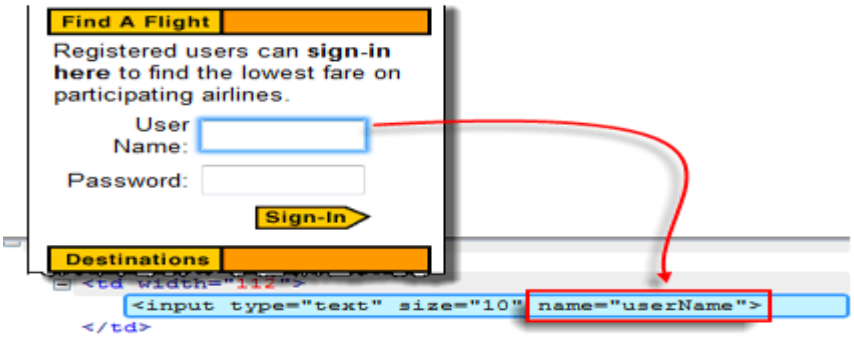

Figure 2.3: Name Locator

iii. XPath Locator

Among the web element locators, XPath locators are remarkably powerful and flexible [21]. It is used to traverse elements and attributes of an XML document. XPath provides various types of expressions which can search relevant information from the XML document. They represent the most general choice because we can specify most locators using properly defined XPath expressions. On the other hand, XPath locators are the most complex selector to use. It requires knowledge in XPath query language, so if we are not fluent in that query language, we will find it difficult to find elements using XPath queries. XPath locator can be an absolute XPath or relative XPath locator.

❖ Relative XPath Locator

For Relative XPath the path starts from the middle of the HTML DOM structure. It starts with the double forward-slash (//), which means it can search the element on the web application.

```
Relarive XPath: //*[@class='featured-box']//*[text () ='Testing']
```

❖ Absolute XPath Locator

The absolute XPath method uses a complete path from the root element to the particular element. An absolute XPath starts with Html and forward-slash (/) as shown below in the example. They are prone to more regression as a slight modification in DOM makes them incorrect or refer to a different element. Normally it's not considered a best practice to use absolute XPath, However, But the risk with this method is, if something changes in the structure of the web application, the code will break. So, this is not a recommended method.

iv. Element with Index

Sometimes there is more than one element present in the DOM with similar attributes and it becomes difficult to search them when they are dynamic. For example, there are 5 buttons on the page and we want to locate the fourth button. Then we search elements with the 'button' tag and navigate to the 4th index of the list of buttons to get that element. These are just a few ways for coping up with dynamic web elements. We will have to use different strategies depending on the situation. The ultimate aim should be to identify a web element uniquely and whatever is the locator strategy, it should always be able to locate the web element irrespective of the change in attribute values dynamically.

B. Visual locator

Visual locators use image recognition techniques to find the element. Third-generation tools have emerged recently. They make use of image recognition techniques to identify and control GUI components. A visual locator has two attributes - a rectangular image and a name. Two visual locators can have the same name. It can implement one "logical entity" which looks different in different circumstances or states by simply creating two visual locators with the same name. Using visual locators is a two-step workflow. The first step is to define the set of visual locator images and their names. Once eyes know what the locator images and names are, the second step is to use an eyes query to search for instances of the locator images on the current application.

A fundamental difference between second and third-generation tools is the way assertions are written. Users can specify fine-grained assertions on the attributes of the web elements contained in the web application's model, or the DOM, using DOM-based tools. Visual tools verify that a picture representing an element's visual appearance is present in the web page as rendered by the browser at runtime. Moreover, DOM-based tools can find the target elements regardless of their position in the DOM or their presence on the displayed GUI. Visual tools, instead, do require the target elements to be displayed on the screen [50].

C. Coordinate Locator

First-generation tools record the screen coordinates of the web application elements and then use this information to locate the web elements [21]. Nowadays this approach is considered obsolete because it produces fragile test cases. Test automation engineers, spend an enormous chunk of test development time finding elements on a page, like buttons, inputs, and div. Finding the correct web elements can challenge, especially when they lack unique ID or class names.

When automating a DWA, the scripts will break as soon as the content changes that will cause the test to fail. Then we have to update the test case each time, which is a tiresome task. Once we understand it, we can devise a strategy to interact with these elements. So, it is not a simple task to handle that element simply by the locator. *e.g.,* in Gmail Inbox elements, their class name gets changed on every login.

2.3.4 Web Element Locators Problem

Before we can configure a better locator, it is helpful to understand why locators failed to locate a web element. As we have stated earlier in Section 2.3.2 rapid development and a high rate of maintenance characterized a web application change to adapt the program for new requirements. Such changes have problems in the web element locator. Choudhary *et al.* [51] define four types of problems that the changes could cause.

The structural change could cause non-selection, mis-selection and form data problems. The non-selection problem refers to the inability of a test case to select a DOM node that was successfully identified in the older version of the web API [51].

A non-selection occurs when a locator L applies to a DOM state D, no items are selected, formally L: D $\rightarrow \emptyset$, but the target element e is still present on the page (e $\in$ D). $L' \mid L'$: D $\rightarrow$ e. The mis-selection problem refers to the selection of an unintended element in the newer version of the web application [19]. The form data problem refers to the use of inappropriate input values within form elements. This problem is caused by the deletion, addition, or modification of form elements. The addition of form elements imposes the use of specific input values for such elements. Hence, the test case needs to be augmented with appropriate commands applying values to the newly added form elements. In contrast, the deletion of one or more form elements requires the deletion of all test script commands involving the deleted form elements. Content changes could lead to the obsolete content problem. This problem is may not be due to bugs introduced in the modification of web application instead it is encountered when the value of the assertion does not match the actual value of the DOM node under consideration.

The element is not found is another common issue with the web element locator. When a user interface element of the AUT couldn't be located or couldn't find an element based on a given locator query, this happened. For example, this problem is caused when *l* attempts to locate an element *e* via an attribute *a* that functioned in previous web application, but no longer functions in new web application.

2.4 Machine Learning

ML is a field of computer science involving the study and construction of techniques that enable computers to self-study based on the input data to solve specific problems [52]. Figure 2.4 depicts the three main categories and subcategories of ML [53]. It is broadly classified as supervised, unsupervised, and reinforcement learning. For each ML approach, one or more algorithms implement the learning approach used to build models for solving real-life problems. ML algorithms extract the internal relationship of the data, which can be presented by some rules or models for prediction and classification problems [54].

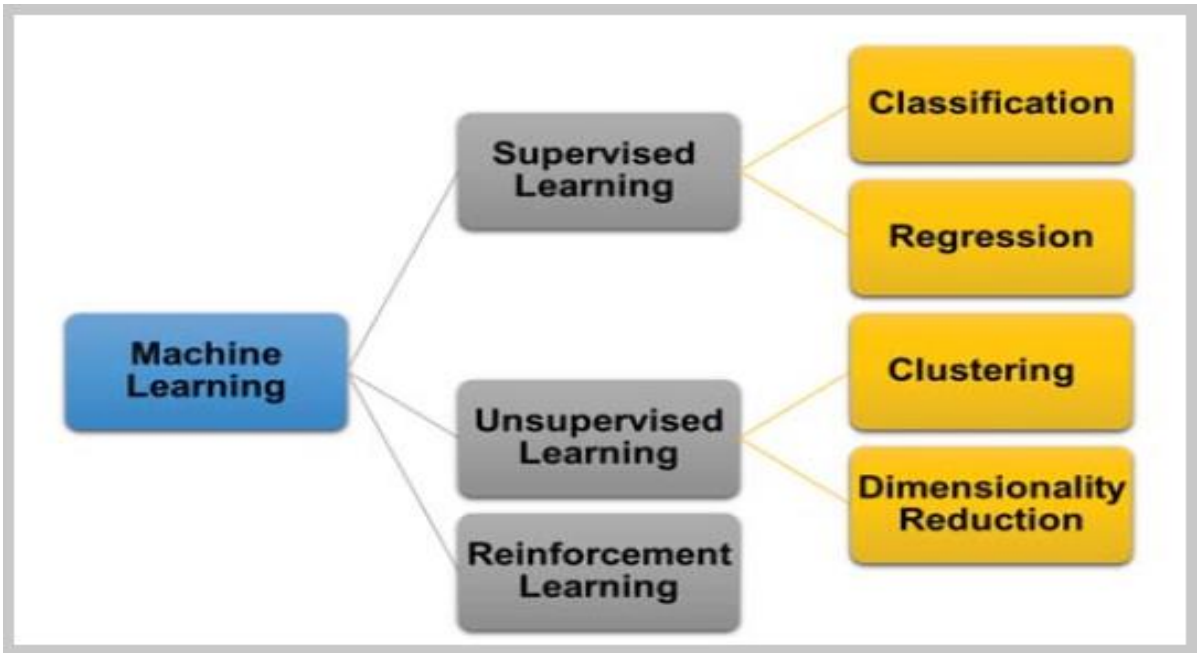

Figure 2.4: Types of Machine learning

A. Supervised Learning

Supervised learning, to make predictions of a set of samples, first provided the training data with all the correct answers for the target variable. The primary task of the algorithm is to replicate the correct response based on the data set to create predictions for a set of samples [54]. In supervised learning, the model requests that you train me. This type of learning is used to achieve AI goals, such as classification and regression. Classification is about predicting a nominal class label and used to predict discrete responses, whereas regression is about predicting the numeric value for the class label and is used to predict continuous responses. This method classifies the free parameters while minimizing output errors [55].

The data set might be separated into training data and testing data, to calculate the accuracy and loss of a particular algorithm.

B. Unsupervised Learning

The ML model has not given any dataset to get trained (no training data) learning through observation and finds structure in the data. This method helps to find hidden patterns in data that do not have a target variable (anticipated outcomes) and organize the data into a clustering system to characterize its structure. It is usually more challenging than the other methods which use for data extraction and clustering purposes [55, 52, 56]. Algorithms used for this method might be k-means where the user specifies the desired number of clusters or hierarchical clustering where the user doesn't need to commit to a particular number of clusters. Both of which attempt to group objects together based on their similarity [57].

C. Reinforcement Learning

This algorithm is constantly adapting to the environment, based on its feedback, which might be positive or negative. This method is concerned with how software agents should take action in an environment [58]. The algorithm chooses an action based on each item of the data set and later learns the effectiveness of the decision, changing its strategy to learn better and achieve the best reward for the action taken.

2.4.1 Machine Learning Classification Techniques

The most popular supervised machine learning classification algorithms are Support Vector Machine (SVM), Decision Tree, Naive Bayes, and K-Nearest Neighbor (KNN) [53].

A. Support Vector Machine

A SVM is a type of supervised machine learning algorithm that performs supervised learning for classification or regression of data groups [23]. The goal is to find the best line or decision boundary which is called a hyperplane. The support vectors are the points where the SVM algorithm determines the line's nearest point from both classes. The distance between the vectors and the hyperplane is called a margin. The hyperplane with the maximum margin is called the optimal hyperplane [59]. The algorithm aims at detecting an object from its background with the minimum training data.

SVM can be of two types:

- **Linear SVM**

Linear SVM is used for linearly separable data, which implies a dataset may be classified into two classes using only a single straight line. The data is therefore referred to as linearly separable data, and the classifier used is known as the Linear SVM classifier.

- **Non-linear SVM**

Non-linear SVM is used for non-linearly separated data, which implies that if a dataset can't be classified using a straight line, it's non-linear data, and the classifier employed is called Non-linear SVM.

Consider a binary linear classification problem with a training dataset of pairings (x_i, y_i) ... (i=1, 2,…n), xi € R^n: and yi € (-1, +1), where x_i, is a feature vector and is y_i a class label. A hyperplane partitions the training data into two sets corresponding to the intended classes in the SVM classifier model. A separating linear hyperplane is defined by equation (1).

$$f(x)\ w^T x+b \qquad (1)$$

where w € R^n and b € R are parameters that control the function. The signed distance between a point *x* and the separating hyperplane is given by the function f. if f (**x**) ≥ 0, a point **x** is assigned to the positive class; otherwise, it is assigned to the negative class [23]. To make the correct classification and get the highest classification interval, the optimization problem of constructing the optimal plane described:

$$\begin{aligned} \min \quad & \varphi(\mathbf{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\left(\boldsymbol{w}' \cdot \boldsymbol{w}\right), \\ \text{s.t.} \quad & y_i\left[(\mathbf{w} \cdot \mathrm{x}_i) + \mathrm{b}\right] \geq 1. \end{aligned} \qquad (2)$$

The optimal solution of **w** and **b** can be solved by introducing the Lagrange multiplier. Then we can obtain the optimal classification problem defined by equation (3).

$$f(x) = \operatorname{sgn}\left\{\boldsymbol{w}^* \cdot \mathbf{x} + \boldsymbol{b}^*\right\}. \qquad (3)$$

B. Decision Tree

The most powerful non-parametric supervised learning method is decision trees (DT) algorithm. The fundamental purpose of DT is to build a model that predicts the value of a target variable by learning simple decision rules based on data features. The root node, where

the data splits, and the decision nodes or leaves, where we get final output, are the two main entities in decision trees [60].

C. Naïve Bayes:

Naïve Bayes is a supervised machine learning algorithm used for classification problems. It is built on Bayes Theorem. It is called Naïve because of its Naïve assumption of Conditional Independence among predictors. The model comprises two types of probabilities that can be calculated directly from the training data: (i) the probability of each class and (ii) the conditional probability for each class given each x value. Once the probability is calculated, the model can be used to make predictions for new data using Bayes theorem [61].

D. K-Nearest Neighbor

The K-Nearest Neighbor (KNN) algorithm is based on the Supervised Learning technique and is one of the most basic ML algorithms. It does not require training; rather, it examines all data points at prediction time and makes a judgment based on the distance between fresh data points and training data points. KNN is a type of classification system depending on its majority, as an object to the nearest class votes to its neighbors, or because it keeps track of all available observations and categorizes them according to their similarity [59].

2.4.2 Feature Selection

Feature selection is a process by which we search for the best subset of attributes or features in the dataset [62]. The purpose of feature selection is to discover the best set of characteristics that improves the classification performance of the models for the problem domain. Feature selection may be categorized into wrapper and filter methods [63]. The filter method selects the features based on various statistical tests to assign a scoring to each feature. The features are ranked by score, and the scores are used to remove low-scoring features from the dataset while keeping high-scoring features [63]. Where as in the wrapper approach, feature selection is ‘‘wrapped’’ in a learning algorithm. In this approach, various subsets of features are generated, and then a specific classification is applied to evaluate the accuracy of these subsets [63]. However, as the space of feature subsets grows exponentially with the number of features, heuristic search methods are used to guide the search for an optimal subset [59].

2.5 Metrics

The evaluation of the ML classifier is important and mandatory to analyze the result very well. To validate the classifier in this research, the researcher selects the dataset (both training and testing) is drawn from each version of a different DWA. The metrics used for evaluating the performance of a classifier are discussed below.

2.5.1 Confusion Matrix

It is a simple performance analysis metrics typically used in supervised learning and used to represent the test result of a prediction model. Each matrix column represents instances in an actual class and the predicted class is represented by the row as shown in Table 2.2.

The four metrics involved in a confusion matrix are discussed as follows:

TP: - Stands for instances that are correctly classified as the instances including correct locator (class actually positive and also classified as positive).

FP: - The instances without correct locators are incorrectly classified as the instances including correct locator (class actually negative but classified as positive).

FN: - The instances with the correct locator incorrectly classified as the instances without fault (class actually positive but classified as negative).

TN: - The instances are correctly classified as the instances without the correct locator (class actually negative and also classified as negative).

When the classifier does things right, true positive, and true negative tells us, while false positive and false negative tell us when the classifier does things incorrectly.

Table 2.2: Binary Class Classification Using a Confusion Matrix

		Actual Values	
Predicted values		Correct locator	Incorrect locator
	Correct locator	True Positive (TP)	False Positive (FP)
	Incorrect locator	False Negative (FN)	True Negative (TN)

2.5.2 Evaluation Metrics

To evaluate the performance of a classifier the most common and well-known evaluation metrics such as accuracy, precision, F1-Score, and recall are used [24]. Each of these is used where appropriate in the analysis of the performances.

1. **Accuracy**: - It is the total number of predictions that were correct or correct classification rate used to measure the proportion of the correctly classified instances to the total instances.

$$\text{Accuracy} = TP + TN / (TP+TN+FP+FN) \quad (4)$$

2. **Precision**: - It is used to measure the ratio of the correctly classified positive instance to the set of positive instances. In other words, members of class members are classified correctly over the total number of instances classified as class members.

$$\text{Precision} = TP / (TP+FP) \quad (5)$$

3. **Recall**: - It the percentage of the correctly predicted positive instance in the whole instance with the defect.

$$\text{Recall} = TP / (TP+FN) \quad (6)$$

4. **F1-Score**: - It provides a good balance between precision and recall, or the F1-Score combines or harmonic mean of precision and recall values.

$$\text{F-Score} = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall}) \quad (7)$$

5. **Support**: - It is the number of actual occurrences of the class in the specified dataset imbalanced support in the training data that may indicate structural weaknesses in the reported scores of the classifier and indicate the need for stratified sampling or rebalancing.

The average of these parameters like precision, recall, F1-Score for individual predictive classes becomes the final values of the entire model such as weighted average. There is no constant formula for a weighted average, but for example, in the case of a two-class problem it can be calculated generally using the equation (8):

$$\text{Weighted average} = (\text{sum of actual class A} * \text{predicted value of A}) + (\text{Sum of actual class B} * \text{predicted value of B}) / (\text{Class A} + \text{class B}) \quad (8)$$

2.6 Machine Learning in Test Automation

Newer technologies and concepts such as cognitive computing, predictive analytics, and ML will be added in automated testing methods to overcome the shortcomings of software testing and make test automation successful [64]. ML aids developers in obtaining accurate outcomes by supporting systems in reading and applying accumulated knowledge. Not to add that the risk of making a mistake isn't the only disadvantage. The amount of data that needs to be processed can still expand without difficulty on the test team, and it also minimized the time required to complete a software test. ML has many techniques and algorithms to be used in software testing [42]. The algorithms and techniques in AI are deferred from each other from different perspectives like how they work, their mathematical and statistical models, characteristics, accuracy when they solve classification or regression or other problems [53]. Many authors use different AI techniques used for many software testing areas [65].

2.7 Summary

This Chapter reviewed various related issues and concepts concerning the fundamental and general definitions, and techniques to DWA testing. especially, web element locators. A web application is a collection of static and dynamic web applications. Web application testing is conducted using manual and automated approaches. Automated testing uses a web element locator to finds and returns web elements on a page using a given query through test scripts. The DWA may change between builds, particularly in the early stages. It becomes a challenging situation for the testers to deal with the rapid evolution of the web application under test. Even minor layout changes may lead to breakages in the test scripts. These changes may affect the test results, or the automated tests may no longer work with the new version of the application.

Automation evolution from recording or linear scripting techniques to page object techniques and describing the pros and cons of each of them. All these methods have a common drawback. Whenever the developer changes any object property, the script execution fails and, we need to spend quite some time to correct these scripts. So, the next level of automation includes a ML technique that detects changes automatically that would navigate the entire application and check if the object properties are the same or changed and need to modify the test scripts accordingly.

3. Related Work

This Chapter briefly discusses major relevant works related to this research work to provide insight into web element locator for dynamic web application testing. Different authors have been proposed variety of web element locator algorithms and techniques. We will try to explain the author's objective, methods, limitations, as well as the results they provide.

3.1 Document Object Model Locator

Dynamic web applications are dynamic and interactive in such a way that their contents and layouts are changed and updated every time as compared to other applications. Therefore, traditional testing techniques and tools are not sufficient for web application testing [66]. The choice of locator depends mainly on the application under test. Many authors have used DOM locators in web test cases to identify GUI objects to operate on and eventually to access web page content in the field of web application testing.

3.1.1 Attribute-based Locators

Attribute-based locator is a type of DOM-based locator. Aldalur and Diaz [15] fix the broken locator’s problem by using web extensions and locate the element using its attributes. If there is no unique node located using these attribute combinations, the attributes of the ancestor node of the elements are used. This process is successively repeated by considering more ancestors until a single node is detected. If no unique node is located, and all the ancestor elements were used, they considered it a failure for selecting the desired web element.

3.1.2 Structure-based Locators

Structure-based locators identify an element based on its position within the DOM tree of the web page under test [67]. Leotta *et al.* [9] proposed the robust locator algorithm (ROBULA) that auto-generates robust web element locators. The intuition behind ROBULA is used to XPath predicates to maintain the locators short and good while retaining a low level of fragility. The locator has proven that these XPath-based robust locators still correctly select the target web page elements when the web application changes. With no experiment for proving these issues, they conclude different locators are adaptable to any web application changes.

The refined version of ROBULA [21], which is called ROBULA+ employs prioritizing algorithms to create robust XPath locators automatically. They address web element locator problems that occur because of a structural change of the web application. The locator selects the nodes in the algorithm starts with the most generic XPath locator that decides all nodes in the DOM. It then iteratively refines the locator until the element of interest is selected. The iterative process degrades the performance of automated testing. The algorithm, as they noted, cannot locate an element after some attributes of the web application changes.

Maurizio *et al.* [43] Instead of depending on ad-hoc algorithms like the one provided by the state-of-the-art utility ROBULA+, formulate the robust XPath locator generation problem as a graph exploration problem. The SIDEREAL tool is the statistical adaptive algorithm, which surpasses ROBULA+ heuristics in terms of resilience by learning the potential fragility of HTML characteristics from earlier versions of the application under test. It significantly reduced the number of broken locators when SIDEREAL used, and the time to construct such robust locators was acceptable. One limitation of this tool is that they frequently produce absolute XPaths that are long and highly flaky.

Aldalur *et al.* [68] proposed an algorithm for repairing structure-based locators called ABLA. They defined four unique attributes: previous ID, previous class, the previous value, and the previous value name used to tag nodes. They calculated the time required to restore each locator by comparing the execution time of ABLA with that of other algorithms that regenerate broken locators and conclude that we get excellent results but have limitations. With ABLA, if the developer has carried out two updates between two executions on the same node in the same attribute, the algorithm will not find the web element because the algorithm only applies to nodes with unique properties. As a result, it cannot apply to all nodes in a web application because all nodes don't have the same properties.

3.1.3 Multi Locators

Leotta *et al.* [69] compared the maintainability of selenium WebDriver test suites employing different locators used in web applications and found that the use of ID locators is much better than XPath locators that depend only on the location of the element in the DOM tree. But when web application developers used auto-generated ID. The ID locator shall be select many web elements or not be able to detect the web element. Leotta *et al.* [46] developed an

algorithm that uses multi locators generated by different tools to select a web element. They gave each of these locators a weight to show its ability to survive among different versions of the web application. If any of these locators could still get a unique element, then the one with the highest weight is chosen. A voting procedure that assigns different voting weights to different locators is the selection criteria. When this procedure succeeds and gets an element identified by the locator with the most votes, other broken locators can repair automatically.

The main motivation behind the use of a multi-locator as opposed to an individual locator is that web element locators are fragile individually, and non-fragile collectively. No research has attempted to address problems related to locator fragility by using a multi-locator approach. Leotta *et al.* [69] address the fragilities of a single locator by creating a novel type of locator called multi-locator, which combines the results produced by different locator generation algorithms and outputs one single locator that corresponds to the most voted one [69]. Multi-locator algorithm is populated with five different XPath generation algorithms, which are used by the following tools: FirePath absolute, FirePath Relative ID-based, Selenium IDE locators, Montoro, and ROBULA+.

3.2 Image-based Locators

The Image-based locators use image recognition techniques to identify and control GUI components. In image-based recognition, an automated test compares a stored image to the screen image pixel by pixel. An image locator has two attributes. i.e., rectangular image and a name. Two visual locators can have the same name. That means we can create a logical object that appears differently in different contexts or states by simply establishing two visual locators with the same name. Sikuli is a tool to automate GUI using the visual image match method. In Sikuli, they should take all the web elements as an image and stored them inside the project. Sikuli will trigger GUI interactions based on the image visual match, the idea which they have passed as the parameter along with all methods [70].

Stocco *et al.* [71] proposed a computer vision-based approach to repair the tests. During the initial execution of the test script, the technique gets visual information (i.e., image capture of web items). They then applied it to identify any changes made to the application in a new version. This method has the drawback of being extremely sensitive to any visual changes in the appearance of a web element.

3.3 Coordinate-based Locator

First generation locators like coordinate locators record the screen coordinates of the web application elements and then use this information to locate the web elements during test case replay [72]. Web element has its position on a page known as X-Y coordinates. X and Y pixels are the units of measurement. X pixels mean the horizontal position of the web element on a page from the left side, The vertical location of a web element on a page is measured in Y pixels [73]. Coordinate-based locates a UI object based on its (X, Y) grid coordinates and a length value. For this to work, the object must always be at the same spot on the screen. Given that any change to the UI layout would break coordinate-based recognition, this approach was fragile even on legacy applications designed for fixed-size screens. It's even more unstable in today's environment of varied screen sizes, resolutions and orientations.

3.4 Web Element Locator Using Contextual Clues

Contextual clues are labels or images near the element that uniquely identifies the web element on a web application. Yandrapally *et al*. [74] proposed a method for uniquely identify web elements based on contextual clues without recording information about the internal representation. They referred to contextual clues positioned in the neighborhood of the web element of interest as contextual clues. The authors use other nearby items to identify the web element of interest. They implemented their method in a record or replay tool called automatic test and analysis qui vicino (ATA-QV) using five open-source web applications in three empirical investigations [74]. The technique comprises two phases: the automation phase and the playback phase. The automation phase creates a test script that records contextual clues for all user interface elements. The playback phase replays the test suite by analyzing the recorded contextual clues and executing the required action(s) on the user interface element of interest. According to the experiments [74] , ATA-QV produces 73 % of the contextual hints that a human could generate. The difficulty with these methods is that ATA-QV test scripts fragile where labels are modified. Such changes could involve changing a label's name, replacing it with an image completely deleting it. If they change the DOM of the web application under test, the playback mechanism can fail. In addition, there is a limit to robustness. Since an eligible locator varies depending on the characteristics of the application, the approach of previously determining the locator lacks flexibility.

3.5 Repair Locators Based on Web Element Properties

Over the past decade, authors and practitioners have proposed different techniques for automated repairing for broken test scripts of evolving software systems [17, 19, 51]. Choudhary *et al.* [51] address test breakages by creating a technique that automatically suggests test repairs. The method consists of two crucial steps, namely collecting test data and suggesting repairs [67]. The data collection phase contains gathering information relative to the old version and the recent version of the web application under test. For every web element in the old and new versions of the web application, ten properties (ID, XPath, class, link Text, name, tag name, coordinates, clickable behavior, visible property, index, and hash value) gathered. They also collect a test script breakage that the error or failure message reported by the testing tool. such information is input to the second phase of the approach. The second phase of the method aims at suggesting repairs for the problems. It generated random values for all the added web elements in the recent version of the web application. If they can find no correct repair, the technique deletes the command at which the test case breaks and checks if the test case passes. Following these steps, the approach suggests a list of repairs to test scripts [51].

Choudhary *et al.* [51] implemented their approach by creating a tool called web application test repair (WATER). They evaluated the method by considering 11 versions of 3 different web applications. The result shows that WATER can suggest correct repairs 81% of the time [51]. The authors do not suggest any method for root cause analysis. Some test cases could present silent breakages, which are situations in which test cases do not explicitly break and keep performing unintended behavior till the end of execution. The authors do not propose any solution to address such breakages. They suggest a repair technique based on random generation of input values for newly added input fields within the web application under test. Random generation of input values may not be effective given that newly added input fields may only accept restricted types of input values. For instance, assuming that the new date of a birth text field is added within the web page under test, such a text field would only expect dates as input values. If the user enters a random string, the test case will fail.

Hiroyuki *et al.* [19], proposed a correct locator recommender approach to support repairing broken locators during software updates. The authors use attributes, images, texts, and

positions to repair broken locators. The suggested method improves the accuracy of selecting correct web elements and focuses on the web applications that vary due to page layout changes. They do not take into account web applications with dynamic content. When there are many similar elements in a page (*e.g.,* web elements in a tabular form, radio buttons with many options), it is difficult for COLOR to differentiate between them. Another limitation of COLOR is that the algorithm will not successfully locate web elements if the developer has performed two updates on the same web element in the same attribute during two runs.

Muhammed *et al.* [20] proposed a model-based approach that is independent of the underlying capture-replay tool. The method targets the repair of test scripts that may break because of locators used in tests, page reloading, and related to pop-up boxes. The model uses DOM-level change information to repair the various test breakages. To prove the generalizability of the proposed strategy, they evaluated an industrial and six open-source case studies. The authors compare their approach with the state-of-the-art DOM-based test repair approach, WATER. The comparison results show that the model-based approach repairs 91% of broken test scripts than the WATER of 85%. The major limitation of this strategy is that it fails when unexpected assertion values occur, i.e., when the textual content of a web element is changed, it requires human involvement to define the functional oracle of the test.

Alshawan *et al.* [75] present an automated technique for regression testing based on the repair of user session data. It contains a white box examination of the structure of the changed web application to detect changes. The authors applied individual URL repair and sequence repair to the session data. They consider ten versions of one single web application to evaluate their approach. It is necessary to apply the approach to a variety of web applications to make generalizable conclusions regarding the performance of the algorithm. The contributions of this work are the creation of an algorithm for regression testing through the use of session data repair and presentation of a controlled experiment's results that evaluate the effectiveness of the technique when applied to ten versions of an open-source web application.

Andrea Stocco *et al.* [76], proposed automatic page object generator for web regression testing by creating a web element instance for each web element. For all of them, a @FindBy annotation specifying the locator related to the web element. The typical problems for this type of locator technique are changes in the UI that break multiple tests in several places,

selectors duplication both inside and across the testing process. In some scenarios, the automatically generated find expression may be ineffective because it locates multiple elements on the active window or may use dynamically generated attributes, which are unreliable for the later test execution. In such cases, they may need to require adjusting the find expression.

3.6 Machine Learning Technique for Web Element Locator

ML techniques implement mechanisms for automatically inducing knowledge from examples. There has been much work that applies ML to software web application testing. Yu *et al.* [77] proposed a multi-Locator based on ML. This method assigns weights to a different locating tool by cyclically training the locators on a web application to improve the locating accuracy of multi-locator. The authors develop several web application testing tools on a corpus of web applications for which successive versions are available. They use same function (*fun_same (e, e') = true or false*) to determine whether the target element and the unique located element by the locator are the same, then if the target element and the particular element are the same use for selecting the web element for testing the web application. The authors conduct an experiment with three web applications. For each web application, there is one original version and one later version. After analyzing the results in which all locating tools returned with the number of broken locators, they conclude that the correct probability of locating the target element in the new web application by learned-weights multi-locators is higher. However, they argue it needs many experiments involving more web applications that are more complex to verify the result. When using this method to perform experiments, different weights to be calculated according to extensive web applications training, and the training cost has no many advantages.

Nicey *et al.* [24] proposed an approach for every element on a web-based platform using ML and selenium. A three-layer structure web scraper, ML, and selenium were used to achieve the goal. They, train the system to recognize specific web elements, and then determine whether the web element is present or not. It does not give a test result of web elements. They recommend as future work to create an algorithm of web testing for a given URL that gives a test result of all web elements or else testing specifically for each type of web element.

Nguyen *et al.* [23] also proposed an automation functional testing framework named codeless web testing by combining selenium and machine learning techniques. They apply the framework to run genetic test cases to test the functionality of search that is present in multiple web pages. They employ the ML learning model to recognize and predict the search element pattern seen in the HTML data of a test web application. While the process is not entirely code-free, there are some obvious and no reasons they should say codeless testing into automated web testing. Even though solutions for automatically generating test cases for web applications have been developed, most test suites are still created manually. *e.g.*, if they want to test a search functionality of a web application, they go ahead with the search button of the AUT and use that link for testing. The main limitation of this approach is it supports the single model that validates a single page based on provided URL or single validation on the current page. It doesn't iterate over a defined set of URLs automatically. Also, they do not address scenarios in which one single locator identifies two or more elements in the next release of the web application under test and if the developer has carried out testing on the same node at the same attribute, the framework will not find the correct web element.

Amr *et al.* [17] introduced an algorithm adopted with a genetic algorithm to select the write element by decreasing the fragility of GUI tests for a web application by automatically repairing the write element. Population, crossover, mutation, and iteration phases are all part of the algorithm. First, the population initialized a crossover performed among some randomly selected individuals, the mutation should apply to some individuals, and the iteration stopped after a pre-defined number of iterations. Finally, the algorithm selects the best individuals. Then, according to a fitness function to be transferred to the next generation. This process continues until they found a result or for a specific number of iterations. The suggested generic algorithm stops after a pre-defined number of iterations.

Using ML in the web element locator of test automation is due to the flakiness, dependability, and maintenance of test automation. Experiments suggest that ML-based test generation can give a greater level of test automation that is broader and more reflective of human testing than current approaches [12], and that identifying web elements supported by ML is more resilient than existing methods [77]. It requires to modify frequently as per platform or environment, and its entire foundation is the application element. This element leads to either change their GUI appearance at the different tests of the web application.

Table 3.1: Summary of Related Work

Title	Authors and Publication Year	Web App Changes Consider	Solution	Limitation
An algorithm for generating robust XPath locators for web testing	Maurizio *et al.* *(2016)*	Structural	More robust XPath locators, which are less likely to break in a new release of the web app	The authors only consider situations in which all the locators point to one single web element in the next release of the web AUT
Approaches and Tools for Automated End-to-End Web Testing	Maurizio *et al.* *(2016)*	Structural	❖ Prioritization and Blacklisting techniques. The locator selects the nodes in the algorithm starts with the most generic XPath locator that decides all nodes in the DOM	Do not address situations in which one locator does not identify any web element in the next release of the web application
Correct Locator Recommender for Broken Test Scripts Using Various Clues	Hiroyuki *et.al* (2018)	Structural	❖ Combined Locator Repairing broken locators in accordance with software updates using element attributes	Did not work well: When many similar elements in a page. It can't handle page layout change
An Automated Testing on Web Based Platform Using ML and Selenium	Nicey and Tommy *(2018)*	Structural	❖ ML and Selenium Train the system for particular web elements and understands what the test cases needs to be executed.	Test data that was once valid but no longer acceptable in the changed app
A Method of Optimizing Multi Locators Based on Machine Learning	Yu *et al* (2018)	Structural	❖ Multi-Locator ML Assigns weights to different tool by training the locators on a web application	They don't address scenarios in which one single locator identifies two or more element.
Algorithm for Repairing Web-Locators using Optimization Techniques	Amr *et al.* *(*2019)	Logical	An algorithm adopted with GA that selects the best individuals according to a fitness function	The algorithm can't set a stop criterion. It degrades the performance.

Algorithm for Repairing Structure Locators Through Attribute Annotations	Aldalur *et.al* *(2020)*	Structural and Logical	❖ Structural locator The algorithm is able to repairing structure-based locators. The locators based on four attributes	When APP updates on the same node in the same attribute, the algorithm not find the correct web element (Element Not found)
Codeless Web Testing using Selenium and Machine Learning	Nguyen *et al. 2020*	Structural and Presentation	❖ ML based Codeless Framework They employ the ML learning model to recognize and predict the search element pattern seen in the HTML data of a test web application	Supports the single model It doesn't iterate over a defined set of URLs. They do not address single locator identifies two or more elements in the changed web app
Automated Model-based Approach to Repair Test Suites of Evolving Web	Muhammed *et al.* 2020	Structural, logical, and presentation	❖ Model-based approach DOM-level change information to repair the various test breakages	It fails when unexpected assertion values occur.
Sidereal: Statistical Adaptive Generation of Robust Locators for Web Testing	Maurizio *et al.* 2021	Structural	learning the potential fragility of HTML characteristics from earlier versions of the application under test.	frequently produce absolute XPaths that are long and highly flaky.

3.7 Summary

A web element locator is a mechanism for uniquely identifying an element on the web content. Locators are fragile upon web content upgrades. *i.e.*, changes in the layout, the appearance of the content of the web application can make locators stop working. When web content updates, they often need to change the test scripts accordingly**.** In automated web application testing, authors made many efforts to identify web elements. They attempted to address web element locator fragility during automated testing by recognizing web elements using different techniques and generations.

The first generation was based on on-screen coordinates, resulting in fragile test suites when the API evolves. Minimal changes to the web application layout or screen resolution may invalidate the existing test cases, requiring the developers to fix or re-create them from scratch. For this reason, they have become obsolete and replaced by other locators. DOM-based locators are the second generation of locators. Those locators overcome the limitations of the first generation by accessing the web application. DOM, a hierarchical data structure where web elements like anchors and buttons can be by accessing their properties or by navigating the DOM tree through structural-based or attribute-based quires. One drawback of these locators is that they usually generate exceedingly multiple locators. It is necessary to consider the locator should have only one candidate element to identify the web element. An excellent locator is unique, descriptive, concise, and applicable even if the web application changes.

Finally, the third generation of image-based element locators or visual locators has been developed. These locators used an image to find a certain element of a web application during AUT. Tests created using third-generation tools are fragile as well, given that simple changes affecting the visual appearance of a web element could cause the test to break. For instance, changing the size of the web element of interest might cause the test to break. Nonetheless, web element Locator fragility is the major problem that requires additional investigation for the dynamic web content of the successive release of a web application. Web application elements with multiple appearances within the same web application (structural change), web application elements that change their functionality at different times (logical change), and web application that generate dynamic web elements attributes are some of the main issues we will consider.

4. The Proposed System Architecture

This Chapter, presents the overall design of the proposed algorithm for dynamic web application testing. Design considerations, the web element locator algorithm, the architecture of the system, and its components have been explained in detail.

4.1 Design Consideration

The DWA is not quite the same as standard web applications from multiple points of view. Dynamic web applications are much more complex on a technical level. They use databases for data loading and its contents are updated each time the user accesses them using values and parameters stored in the database. This design aims to propose a web element locator algorithm that can correctly locate web elements of AUT. The following are some of the design consideration issues for the newly proposed solution.

A. Accuracy

Finding the right web elements is a vital requirement when creating an automation script. The web element locator algorithm will locate the write element. The proposed algorithm will test whether it improved the accuracy of locating the write web element for DWA testing and compared with other related algorithms. It is important to have good locators to find the correct web elements, making sure that tests are faster, more accurate, and more maintainable.

B. Test Maintainability

Test maintainability is performed to evaluate the software or system's ability to get modified without any issues, to satisfy and meet the changing requirement and needs of the user. This type of testing ensures the hassle-free implementation of the modifications in the system [78].

The proposed design considers the web element locator problem in DWA testing which is occurred during the evolution of web applications. The algorithm tries to adapt to the change of web application and correction automatically handle the web element change to continue the test, it will reduce the maintenance effort. The maintainability of a test can also be measured in terms of the effort required to test the changed web application. It can be a correction of errors or faults and adapting to the changing environment. Tests should integrate

with such pipelines and automatically execute when triggered. For faster execution times and mitigate testing problems, the proposed algorithm integrates the change every time.

C. Execution Time

Execution time for regenerating a broken locator is an important factor for successful web application testing [2, 68]. The process of locating the correct web element and restoring the broken locator takes longer duration. Even if we use ML, we will try many features to kept the ones yielding the good performance and moved them to the final model. The quicker the test case completes, the better it is. The execution time of the proposed algorithm depends on the number of target elements.

4.2 System Design

This section presents the architecture and the components of the proposed algorithm for DWA testing. In Figure 4.1 overall proposed algorithm architecture is depicted. Web applications follow the n-tiered structure by nature, and every tier has a designated role even though there are other variations [29]. The first tier, known as the client (presentation), is represented by a web browser. This thin-client web browser is mainly responsible for viewing and rendering web application documents on-screen and generating requests to a web server. The second tier is the middle tier which is the web application itself, running a dynamic web content technology. The dynamic web content technology that is the application server is responsible for generating HTML content that is sent back by the web application server to the client. The third tier is the storage tier which is responsible for persisting the data, normally using a database.

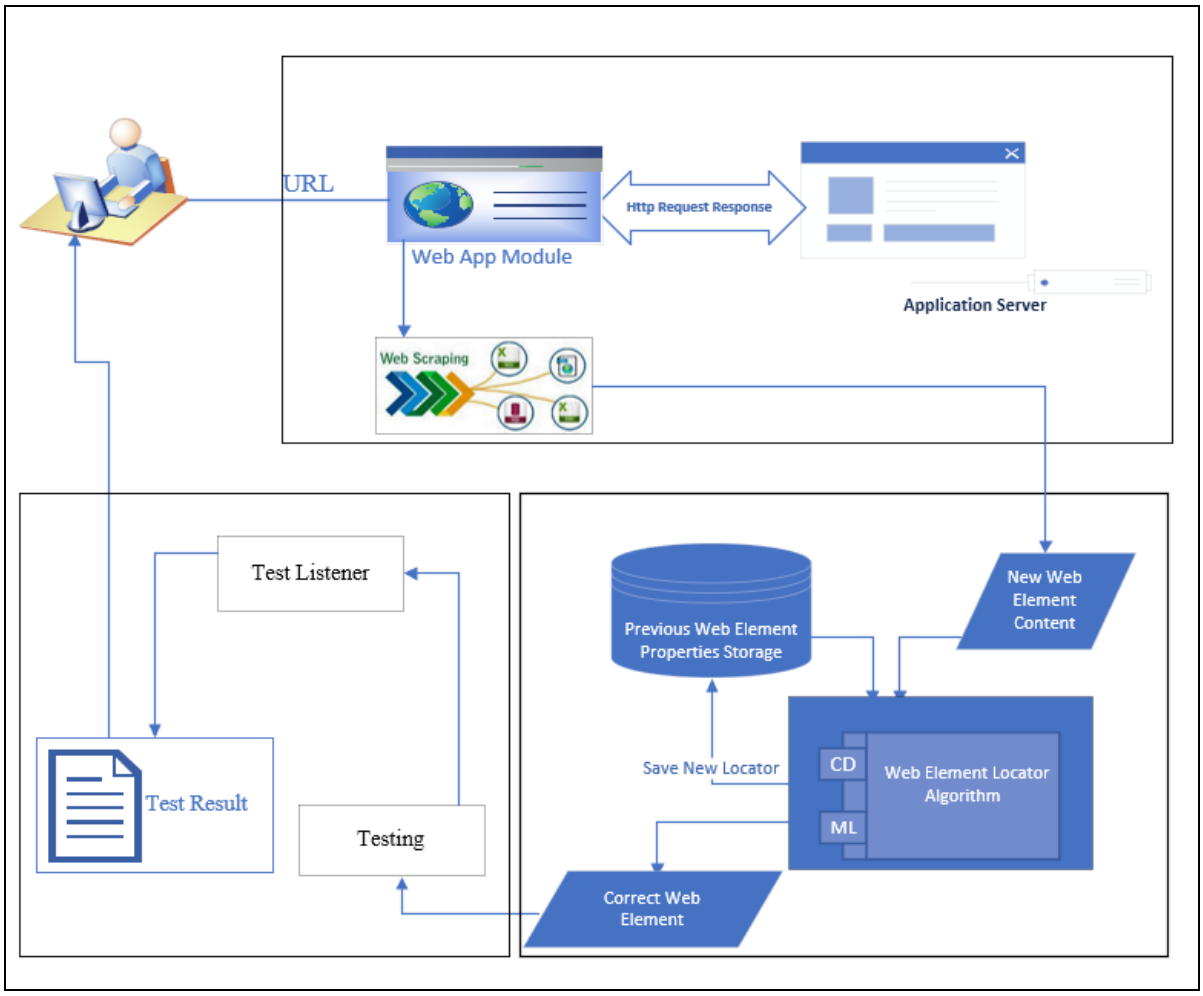

Figure 4.1: The Proposed Architecture for Dynamic Web Application Testing

4.3 Description of the Components of the Proposed Algorithm

A. Web App Module

The web application module accepts the address of the DWA to be tested and crawls through the web application to retrieve all elements.

B. Application Server

The application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb *etc.* It is a component-based product that lies in the middle-tier of a server-centric architecture[1]. An application server is software that helps a web server process the pages containing server-side scripts or tags. The application server passes back to the web server,

[1] https://www.javatpoint.com/server-web-vs-application

which then sends the page to the requesting browser. All the browser gets when the pages arrive is pure HTML.

C. Web Scraping

We extract important information from the web application elements and their position in the DOM tree with their attributes. After scraping data from the web application, the data clean up and extract from the given data that can be used for the next stage.

D. Previous Web Element Properties Storage

Now, we postulate that there are web element properties storage that contain a web element locator and web element properties for the previous web application. The web element locator is not executable for the changed (new) version of the dynamic web application. A simple modification of the API test has an impact on locators that leads to unable to select the desired web elements. It is used as input to the change detection phase of the web elements locator algorithm, that next time learns the change of each web element throughout the process.

E. Testing

In web applications, web tests are concerned with performing actions on certain elements in different pages of the application. *e.g.*, finding a button to click on, finding a text field to write in, *etc.*, and comparing the output of these actions with a predefined expected result. To perform testing on any element, the element must be firstly located. Based on how test cases localize the web elements to interact with, what kind of objects select the target web elements.

F. Test Listener Module

As the name implies listener module listens to every event that occurred during testing and behaves accordingly like test start, test finish, test fail, and test success. Test listeners watch the test behaviors and to allow customizing test reports or logs. Each Listener contains different methods invoked to perform a specific task. For instance, when a test case fails or is skipped, then want to take a screenshot of that particular automation scripts to see what went wrong. In that case, it can invoke the test failure method within the test listener and move the test execution to a script repository and execute a specific event. A good practice is to log a line of text before every action. This log will be displayed in the report and a will know which action the test case has failed.

G. Test Result

Clear and comprehensive reports help us to conclude after script execution is complete. The test result should be able to understand what it represents, the test coverage, the pass percentage, and the overall state of the DWA under test. Good test execution reports are essential to understanding the state of the application under test. Especially if the number of tests is large. The test result saves in HTML files, for human-readable results.

4.4 Web Element Locator Algorithm

Web element locator algorithm contains change detection (CD) and generate a new locator for the changed web application based on the change using ML. The CD is based on the comparison of the previous version or the web application before the change and the current AUT or the web application that changed due to modification or update.

4.4.1 Web Element Change Detection Algorithm

The web element CD algorithm accepts the application that is going to be tested which is selected by the users. After that, the algorithm then compares the two-web application to detect the changes. It first selects the previous web application and, then the modified web application as an input. Then constructs the parse tree of both the pages and then compares the node by node of both the trees. If the web element changes and the locator is not aware of the changes then it will be looking for an element that may or may not exist.

Web element change is detected by combining these two different detection algorithms [45, 20]. In our context, it requires a more detailed comparison of the various DOM elements and are interested in all the changes that may cause a test breakage. These changes include structural, logical and presentation change of web element.

The CD algorithm consists of three successive phases. The pre-processing, processing, and post-processing phase. The pre-processing phase is responsible for converting the initial web application into element queues that are used for further analysis. The processing phase receives as an input of two web applications for checking the application change based on a comparison of two web components. In this phase, the checksums are calculated for the whole DOM object and the DOM object's internal content. When the data extraction is complete and the checksums are calculated for the DOM object, the DOM object's data and its uniquely-

generated identifier are used to create a unique node that is appended to the tree as shown in Figure 4.2. An element is formed from a unique identifier of the DOM object (extracted from the tree), a unique parent identifier of the DOM object), the weight of the DOM object, and a sub-tree that contains the DOM object itself and its related first-level children.

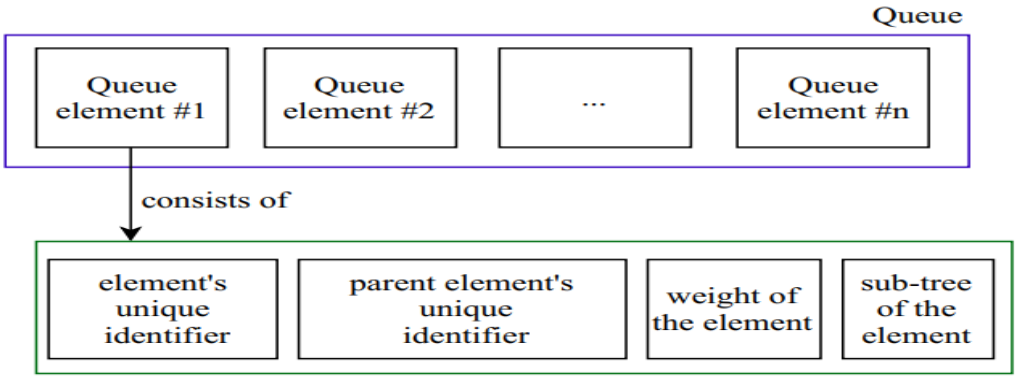

Figure 4.2: Generated Element

The post-processing phase is the final stage of the proposed detection algorithm's execution. The basic idea behind this phase is to group components with relevant statuses from CD queues.

Different changes are made during the development process like structural change, logical change, and presentation change. It can impact the test script, for example when new functions are added after the test script was generated it must update the test script to automatically locate new elements. The same is true for the deleted element also, the test script values must be deleted corresponding to the position from where the web element was deleted. Figure 4.3 explained details the change that occurs in DWA changes and how to adopting it.

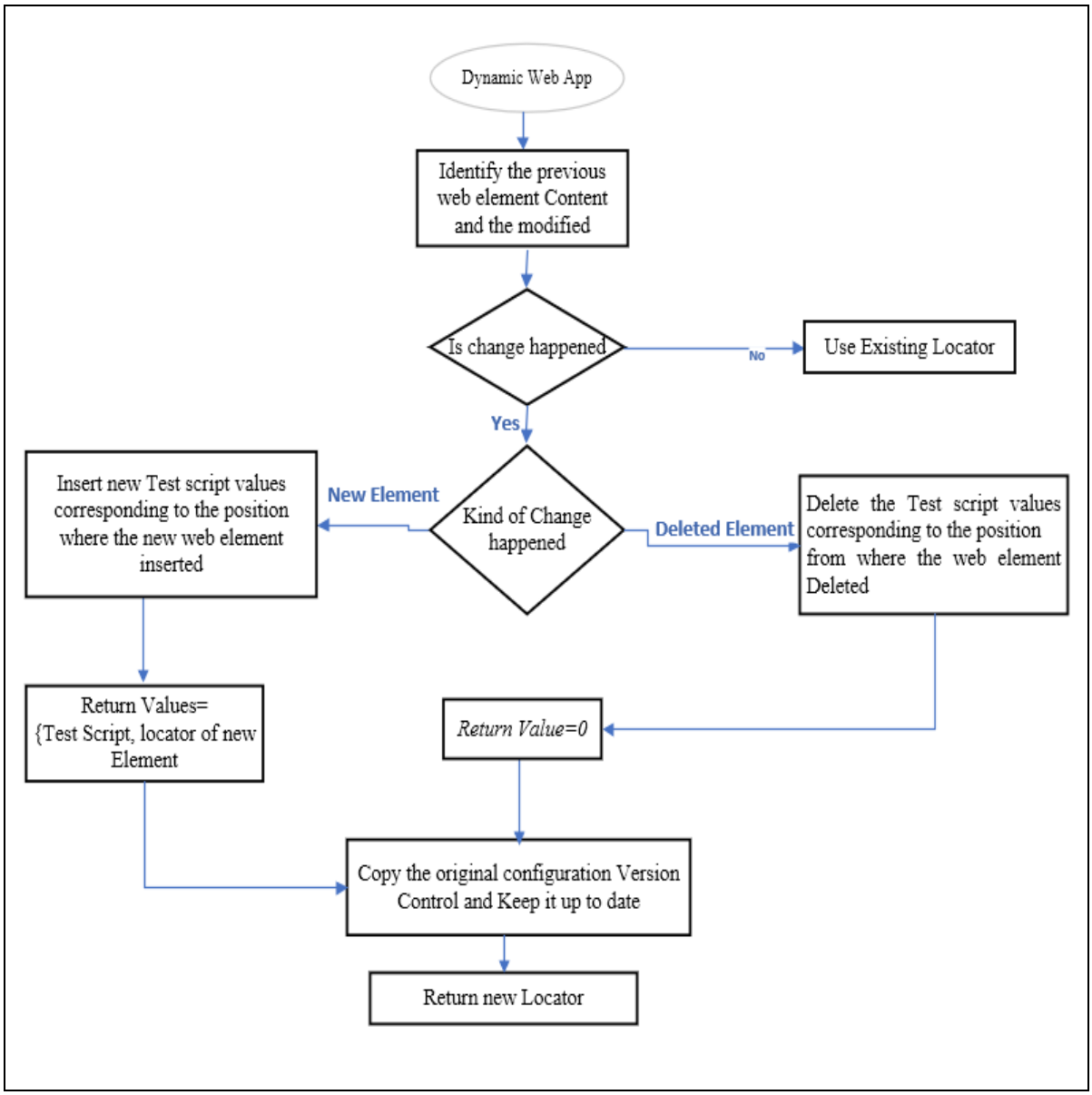

Figure 4.3: Change Detection

To identify web elements WELA learns the change throughout the process for each web element from the previous web element properties. Generate dynamically access alternative locator is part of the algorithm. That means, propose different locators in case of failure dynamically access alternative locators and proposed automatically try and re-execute based on previous experience.

We scrap the web application of the first execution or previous version of the web application and the new changed web element after the web application changed and match their respective web element. The matching web element or elements that have the same path or

properties are identified as shown in Algorithm 4.1. The find_best_matching() function determines which attribute should be selected for identifying the target element The non-selection of web element problem and the mis-selection problem are addressed by comparing new and previous web element properties of the web application under test. If any of these properties match, an updated locator is proposed as a solution. The comparison of elements' internal contents is done by applying Levenshtein's distance algorithm which provides a percentage value between two web elements. if there should be an occurrence of any change at the web element, we use extended find features of the web element using the ML model to generate a new locator for a needed web element.

It will find a similar pattern of the web element and adjust it according to the changes. Finally, new web elements are identified. The next step is generating a new locator for given web elements.

```
   Input: Old_ element_T1 and New _element_T2
   Output: The matching element
Function Document_Root_Analysis (New Element, Old_DOM element)
    find_best_matching(Attribute)
     new_element_T2_Path← GetPath(new element T2)
     old_element_T1_Path← GetPath(old elemen T1)
  if new_element_T2_Path=old_element_T1_Path then
        new_element_Attribute ← Get Attribute (new_ Attribute)
        old_element_ Attribute ← GetID (old_ Attribute)
  if new_element_ Attribute = old_element_ Attribute then
      new_element_ T2_weight ← GetWeight(new_root_element)
      old_element_weight ← Get Weight (old_root_element)
  if new_element_weight = old_element_weight then
new_element_T2_checksum← getDOM Checksum (newroot_element)
old_element_T1_checksum← get DOM Checksum(old_root_element)
   if new_element_ T2_checksum = old_element_checksum then
    calculate LevenshteinDistance (T1. weight, T2. weight)/max (length
                    (T1. weight) length (T2. weight)
                fun same (T1, T2) =1
               return true
                     else
             return false
          end if
   end function
```

Algorithm 4.1: Algorithm for Web Elements Change Detection

4.4.2 Machine Learning Model

Choosing a model can be achieved by comparing accuracy, training time, and prediction time for different combination of feature extractor with classification models. According to [24, 79], a linear support vector machine can often classify a dataset to higher accuracy. It was a benchmark with other algorithms such as decision tree, fine decision tree, logistic regression, Gaussian Naïve Bayes, and Kernel Naive Bayes. Based on the advantage that the linear support vector machine classifier algorithm provides, we used SVM to build and train a model in this research. As discussed in Section 2.4.1, SVM belongs to the linear classifier family of supervised ML algorithms that attempt to find a linear hyperplane that separates examples from different classes.

For any change in the web element, the SVM is to train a model that assigns a new changed web element to find a similar pattern based on those features to adjust the locator according to the change. Figure 4.4 shows the detailed process of the ML training model for locating web elements. It includes two major steps: the first step is model training, and the second step is locating the write web element.

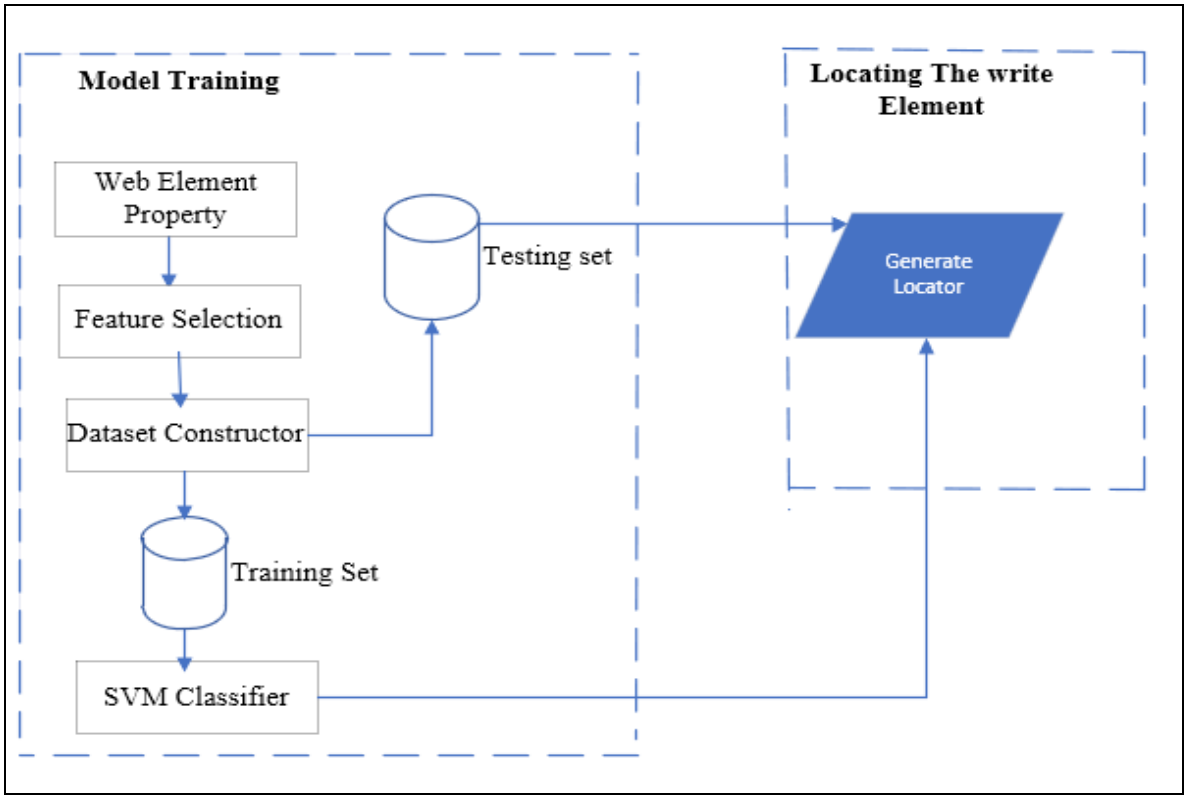

Figure 4.4: Machine Learning Web Element Locator Model

4.4.3 Feature Selector

A significant feature of web element which is useful for algorithm is extracted. Extraction the feature of individual web element properties being observed once structured data is extracted for each DWA element. Many features and preprocessing steps as part of the identifying the web element process, like capture the element, capture the element information or features, and feature synthesis as shown Figure 4.5. Features are extracted from the element node, containing information about element node name, element node type, element node values, and, the element node attribute. The node type property of the DOM is very helpful in determining exactly the type of node currently accessing, which is not always so apparent and the node Value property is a read or write property that reflects the current value of the element node. All web element properties might not equally contribute to the classification result of the ML model. Through feature selection, a significant subset of features in the dataset is selected.

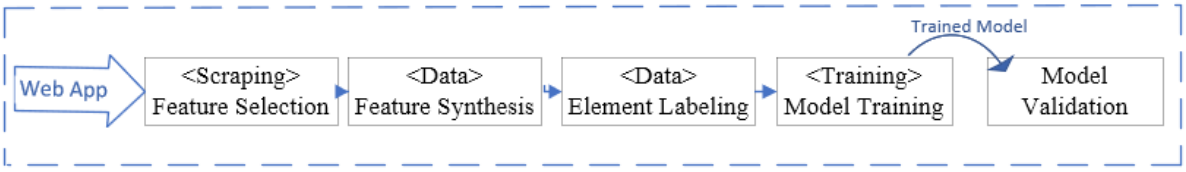

Figure 4.5: Feature Selection

4.4.4 Dataset Constructor

A large high-quality training set is required to develop a ML model that can help the algorithms to understand that certain patterns or series of outcomes come with a given question. The dataset constructor is used to convert the comma separated values (CSV) file into sets and the data is divided into training and testing datasets. The constructor will first read the CSV file with Pandas, data manipulation and analysis tool built on top of the Python programming language, before passing it to NumPy arrays. NumPy can process numbers, texts, and records as arrays. Finally, the dataset constructor split the dataset into training and testing data and then perform data conversion.

A. Splitting Dataset

The training dataset is used to train the algorithm to develop a model or set of rules that can predict outputs based on input variables whereas, the testing dataset is used to evaluate the model's classification performance. To split the CSV dataset into training and testing sets, the dataset constructor uses Scikit. Scikit library contains built-in classes for different ML algorithms that offer various features for data processing that can split a dataset into random train and test subsets. According to Gholamy [80], to get the best results when 20-30% of the data is used for testing and the remaining 70-80% for training. In our algorithm also use 80% for training and 20% for testing as shown Annex A.

B. Data Conversion

ML models cannot be used the CSV file directly, it needs to be converted the input and output variables to be appropriate format. This means that if the ML data contains categorical data, it must encode into numbers before it can fit and evaluate a model. To do that, we create a list of numerical and categorical features, respectively. Next, pack the numerical features into a single vector and the categorical features into vocabulary list. We used a Label encoding for converting labels into numeric values. to make them understandable for machines. In general, the features of ML data can be:

- **Numerical**: Features whose values are integer value or continuous. They are represented by numbers. For instance, feature "locator*_label"* whose value is a binary set: {0 or 1}.
- **Categorical:** Features whose values are taken from a defined set of values. For instance, feature "*Web element Attributes Properties"*: {Name, Id, role, class, arial-label, accesskey, position} is a category value.

4.4.5 Classifier

Once all of the data has been labeled, we can now make use of machine learning classifiers, for the given problem. We discussed many supervised based ML classification techniques earlier in Chapter 2 under Subsection 2.4.1. Among the techniques, we are going to use the classification technique named linear SVM to build our model. Dynamic web elements are keep changing dynamically in structure, logical and presentation. In our proposed algorithm, the objective of Linear SVM model is to recognize the pattern of web elements. For any

change in the web element, the SVM linear classifier that identifies similar patterns to a newly changed web element based on the web element feature. Finally, the ML model generate a new locater to automatically identifies the web element.

4.5 Algorithm for Dynamic Web Application Testing

WELA is applying for DWA testing. The algorithm can automatically identify the change made to an element locator that was added between predefined test automation steps, and either quickly fix them. Whenever locate a web element, it is very important to analyze what remains constant and what changes with every application. Once this understanding has, then it will be able to create many reliable locators.

For simplicity and better understanding, WELA is split into three phases as shown in Algorithm 4.2. Get the web AUT (phase 1), identifying change and locating the web element (phase 2), and performing web application testing (phase 3). The following processing model must be followed to determine what the indicated part of the algorithm follows to locate a given web element and execute automation testing.

1. At lines 1 and 2 of the algorithm, parse the AUT and allow the component of the Application to be identified.
2. Starting from line 3 up to line 10 of the algorithm, If the new test execution finds an element in the DOM with an attribute and content equal to the web application that ran earlier, then that web element is the target element. It shows that the web element remains unchanged, the locator remains correct, and the algorithm continues to run.
3. When the locator cannot select the target element, an element is not correctly located, and it generates an error message, showing locator failure. Then the CD phase of the algorithm detects the changes of the web elements.
4. Identify useful features of individual dynamic web element properties being observed and apply Linear SVM. The machine is trained to locate the web element based on the collected information and returned the selected web elements.
5. If the processing part of the matching algorithm is performed successfully, meaning that by generating a new locator the target web element is identified.
6. Once the web element is located, the final section of the algorithm is performing a web application testing, then the results are analyzed and defects are logged, if any.

Input: AUT

Output: **Test Result**

Definition: Let AUT, be a dynamic web application, T1, T2: Dom tree collection for previous web element and the new or the changed web element,

```
L: Previous locator, L'=new locator, mseg,:Error or failure message
  //phase 1
1.    page = requests_get (AUT)
2.    soup = BeautifulSoup(page.content, DOM.parser')
3.     for locatorList L do
4.     attribute=get LocatorAttribute(L)
5.     Value=getLocatorValue(locator value)
6.     Path=getLocatorPath()
7.     Element=find_element(L)
8.If (Element=found) then
9.    Locator L correct <- correct value
10.   execute Test ()
  //phase 2
11.else if (Element= "changes") then
12.       mk[]= element.getChanges()
13.     features = getAttributes[T1])// find previous reference
14.     element = self (element, features)
15.     match Features (features T1, features T2)
16.     foreach.add(getElementByFeatures(T2, locator T1)
17.      foreach (T2 in new element) do
18. If mk[]=logical change
19.     If mk[]== new Element()
20.      L'.find elementsBy(features)//generate new locator(L')
21.       Find changed L'= updated L ()
22.        Element=find_element(L')
23.      Else mk[]==deleted Element ()
24.        Remove (element)
25.  else if mk[]= structural change
26.         L'.find elementsBy(features)
27.         L'= Update.attribute(newValue) //update attributes
28.  else mk[presentation change]
29.        L'.findelement (Contains, starts-with]// stable Locator
30.        locator=locator L'// update locator
31.         Element=find_element(L')
// phase 3
32.      execute Test ()
33.      return result ()
34.    return false
35.else
36.   return false
```

Algorithm 4.2: Algorithm for the Web Element Locator for Dynamic web App Testing

4.6 Summary

The Chapter systematically went through the development of a web element locator algorithm for DWA testing. The components that constitute the system, the relationship among these components and the responsibility of each component are presented. The designed system encompasses many components working in collaboration to perform different tasks like scrapping, ML model, locating web elements, identifying web elements, and executing web application testing.

We divide the WELA algorithm into three phases. The first phase is getting the address of the AUT that is going to be tested. The second phase is identifying change and locating the web element. The change identification is based on the notion of feature, which reflects the degree of similarity between the changed web element and the previous web element. A linear SVM classifier is used in the proposed ML algorithm to generate a new locator for the changed web element. The best set of web element features allows building useful ML models. The last section of the algorithm is doing web application testing after identifying the target web elements.

5. Experiments

This Chapter, described the implementation details of the algorithm for DWA testing based on the theoretical basis described earlier. The development environment and the tools used to develop the system are briefly discussed. Accordingly, the type of ML classifier, the data set used, and the results achieved in the linear SVM process will be discussed. The implementation details and evaluation of the proposed system are also explained. Finally, the results of the algorithm for the problem domain with another related algorithm will be discussed.

5.1 Tools and Programing Language

Various tools and libraries used for the development of different components the algorithm. The tools, libraries and languages with their respective description are discussed in Table 5.1.

A. Selenium

We used selenium automated tool for testing WELA. One of the core ideas and motivations behind the use of the selenium tool is that according to the review by Gallego *et al.* [81] selenium tool is one of the top automated software testing tools in 2021 for web applications across different browsers and platforms. One of the major components of selenium is the selenium web driver, a friendly API that makes tests easier to read and maintain. Currently, selenium supports various programming languages that can be written in Java, Python, C#, Ruby, Perl, and .Net. This allows a large number of testers to easily use it without any language barriers.

B. Python

ML applications and ML algorithms are written and designed using programming languages Python is extremely popular in the developer and coding community. It is a highly dynamic, open-source language that supports object-oriented, imperative, functional, as well as procedural development paradigms[2]. We used python 3.9 with PayCharm 2020. PayCharm is a cross-platform IDE used for Python programming[3]. It is one of the Python IDE editors that can be used on Windows, macOS, and Linux. This software contains an API that can be

[2] https://www.upgrad.com/blog/best-programming-languages-for-machine-learning/
[3] https://www.guru99.com/python-ide-code-editor.html

used by the developers to write their Python plugins so that they can extend the basic functionalities. To perform data preprocessing in ML using Python, then it needs to import predefined Python libraries[4]. These libraries are used to perform some specific jobs.

C. Beautiful Soup

According to Uzun *et.al* [82], there are several libraries for extracting useful data from web pages in python. Uzun *et.al* compared three different well-known web content extraction libraries including BeautifulSoup, lxml, and regex. They conclude that regex achieves the good results with an average of 0.071 Ms. However, it is difficult to generate correct extraction rules for regex when the number of inner elements is not known. So, for this research purpose, BeautifulSoup and lxml are used for the extraction process.

Beautiful soup is a Python library to crawl web applications or screen scraping AJAX-enabled websites[5]. All kinds of information extract from the web application elements and their position in the DOM tree with their attributes. After scraping data from the web application, the data clean up and extract from the given data that can be used for the next stage.

D. Fuzzy Wuzzy

Fuzzy wuzzy is a library of python which is used for string matching. Fuzzy string matching is the process of finding strings that match a given pattern. It uses Levenshtein distance to calculate the difference between sequences. It is used for to calculating the similarity between the new and the previous web element.

Table 5.1: List of Development and Experimentation Tools

Name of Tools and Libraries	**Description**	**Version**
Selenium	A tool that used for testing web application	3.141.0
Python	Python is a high-level general-purpose programming language. It is used it to write the program (code).	3.9.1
Beautiful soup	Extracting useful data from web application	4.9.3

[4] https://www.javatpoint.com/data-preprocessing-machine-learning
[5] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Fuzzy Wuzzy	It is a library of python which is used for string matching	0.18.0
NumPy	Support for large, multi-dimensional arrays and matrices for converting the array of int labels into a NumPy array	1.20.3
Scikit-learn	Built-in classes for different SVM algorithms used for classification purposes.	0.24.2
Matplotlib	Plotting library for the Python	3.4.2
Pandas	Manipulating numerical tables, time series, and reading files	1.2.4

5.2 Experimental Setup

In this research, the experimentation was conducted using a computer with Intel(R) Core (TM) i5_4200U CPU of 1.6GHz processor speed and 8.00GB (RAM), 1000 GB hard disk capacity with window platform.

5.3 Data Sets

ML techniques require representative data, possibly without bias, to build an effective ML model for a given problem. To test dynamic web applications using the proposed algorithm i.e., WELA, needed a fair number of web element for training purposes. Five thousand two hundred eighty web elements collected across 250 web applications with the help of beautiful soup from SourceForge and Alexa database. Each element in the datasets is evaluated using a combination of seven attributes (ID, name, title, class, role, aria-label, and access key) as shown in Annex B. The stated attributes were selected based on the study addressed [17] which prioritize their ability to successfully locate an element among different versions of an application. All the web applications that were used are legal and included a variety of web elements. It needs to employ the processing step in the next steps to improve data quality and extract web elements relevant to the research.

5.4 Implementation Process

This section will demonstrate how to execute a testing process from the beginning to the end of the testing with sample pseudocode. An experiment is performed on a sample of ten open-

source web applications from SourceForge, and common example of dynamic applications such as Amazon, Gmail, and Google Search as shown in Table 5.2. The applications are relatively new and well-known, with at least two major releases to ensure that they work properly on the most recent versions and some of them have been downloaded more than one hundred thousand times.

Table 5.2: Sample Web Application

Web Application	Original Version		Modified Version		**Test Script**	**Web Application**
	Line of code (LOC)	Version	LOC	Version		
Password Manager	112,010	0.3.1	575,976	0.6.0	42	http://sourceforge.net/projects/ppma/
University management	743,624	1.2.0	1,055,988	1.5.0	250	http://sourceforge.net/projects/claroline/
Room Booking	11,839	1.26.1	34,144	1.4.9	35	http://sourceforge.net/projects/mrbs/
Phone Book	10,338	4.0	34,548	8.2.5	55	http://sourceforge.net/projects/php-addressbook/
E-learning	352,565	1.10.7	333,551	1.11.5	53	http://sourceforge.net/projects/claroline/
Bug tracking	108,473	1.1.8	180,741	1.2.0	49	http://sourceforge.net/projects/mantisbt/
Collaboration Software	149,887	0.65	220,993	1.0.0	44	http://sourceforge.net/projects/collabtive/
Google						https://www.google.com
Gmail login						https://www.gmail.com/
Amazon						https://www.amazon.com

Various scenarios are investigated where WELA can provide a significant amount of improvement in terms of time and effort required as well as the accuracy of locating dynamic web elements during web application testing: A detailed evaluation of WELA under these scenarios is provided in the following section. The scenario is related to the changes of web element that is logical change, structural change, and presentation change of a DWA and address the web element locator problems like non-selection, mis-selection, a form data problem, and an element not found problems. To test the algorithm, we defined three conditions on the result of the test case: pass, fail, and error.

- The test is considered as pass, if a web element locator algorithm identifies web element correctly. For example, to test out a login feature locate an input text field, fill it with some text, locate a button, click on it, locate the text that shows the result of the computation and check whether it matches the expected behavior of the web application. Finally, the test result of the web application displayed correctly as shown in Figure 5.1.

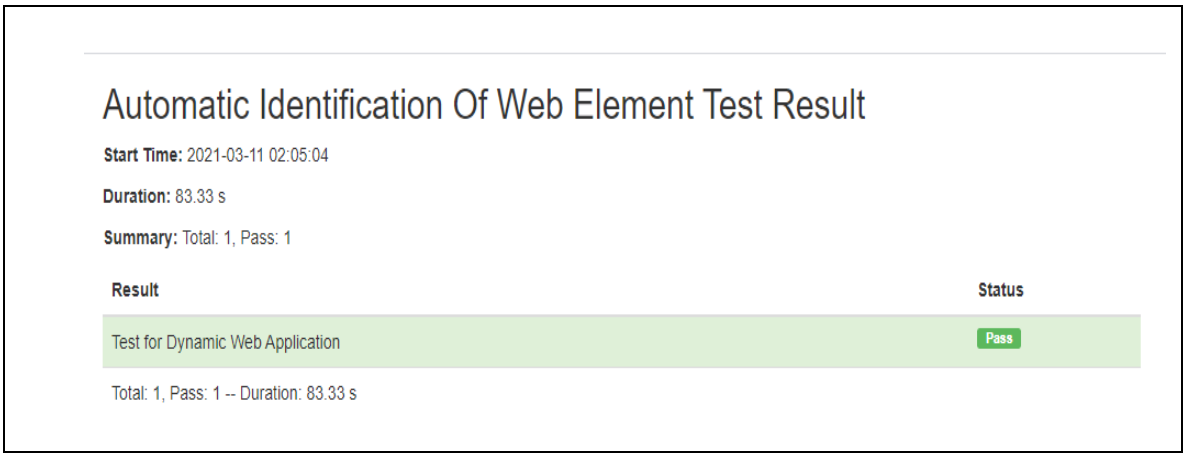

Figure 5.1: Pass Test Result Example

- When the web element changes (dynamically generated i.e., new elements added, elements deleted, elements move) and the locator does not select those elements the test is considered as fail as shown Figure 5.2. It leads to the mis-selection and non-selection of web element happen and the result of the test is not as expected.

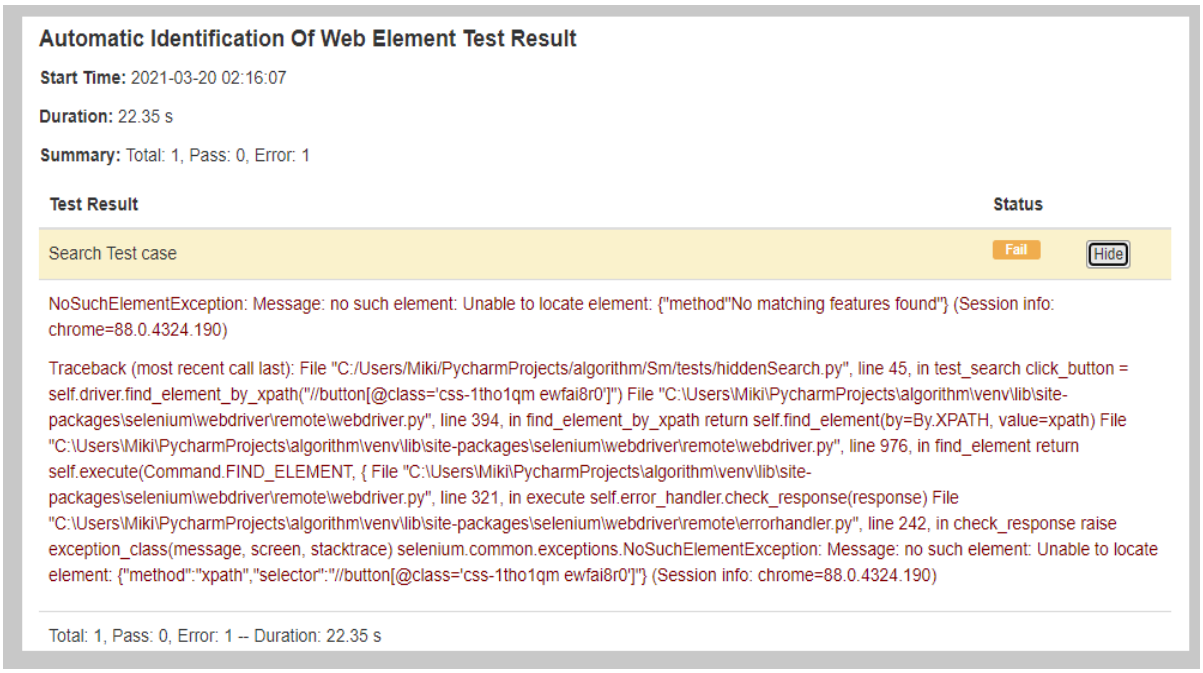

Figure 5.2: Fail Test Result Example

- Error criteria are considered when an error occurs while running the test that is not related to the objective of the test. *e.g.*, some test scripts fail when the application takes too much time to load the user interface. Another type of error is related to user session timeouts. This problem occurs when the web application becomes inactive during testing after *n* minutes of inactivity as shown in Figure 5.3. For example, a web application has a preset session timeout of 10 minutes while the test suite running on this application takes almost half an hour to complete its execution. During testing when the application logs out after 10 minutes, the test scripts error.

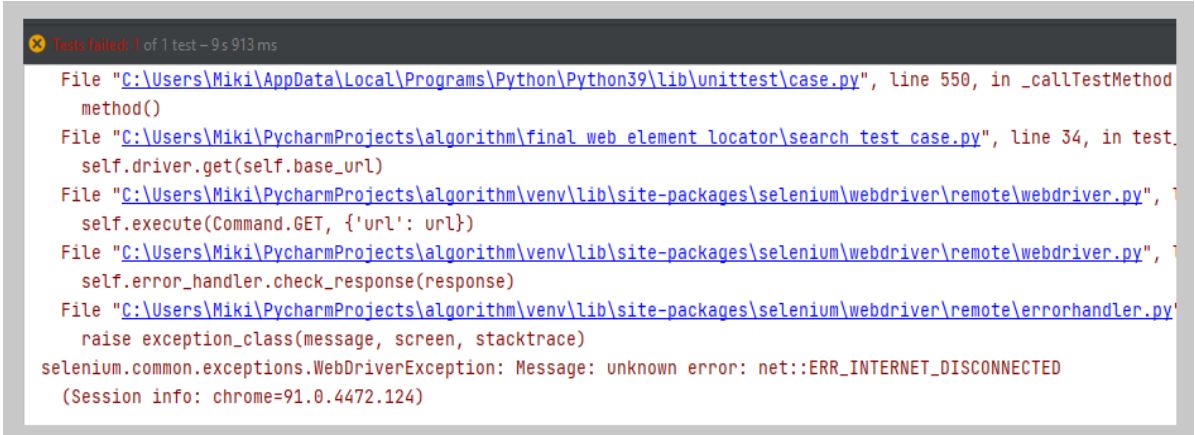

Figure 5.3: Error Test Result Example

A. Logical Change of Web Application

Under this scenario, the main focus is on modifying the logic of the web application under tests such as functionality addition, modification, and deletion. Then check whether the algorithm has overcome this problem, which means if an object gets modified or removed then the program script must be able to find the closest object to the one it was looking for and decides the other locator strategy to locate a particular element.

Consider the following example that is to test out a search feature. The tester runs the web application and the search page is displayed. The first attributes that are shown are the search by price and search button as shown in Figure 5.4. Imagine there is a new version release and the advanced search option was added or a logical change happen in the old web application as shown in Figure 5.5, so the search functionality locator has changed.

WELA will not be deceived and will automatically overcome these changes, in this case, the search page gets modified, then the program and the script must be able to find the closest object to the one it was looking for and then generate a new locator for the advanced search option and update the existing locator and continue the test. i.e., the search page has the options, search by price, search text box and search button first, and the script was written according to that, and for the new advanced search option the script must be able to add new test script to the existing one using create new test command corresponding to the added field and anticipate that change so that the test run can continue running without fail. If a web element is removed the same is true that the algorithm uses the removed test script command and removes the test scripts that are used to locate the removed element.

Figure 5.4: Old Search Functionality

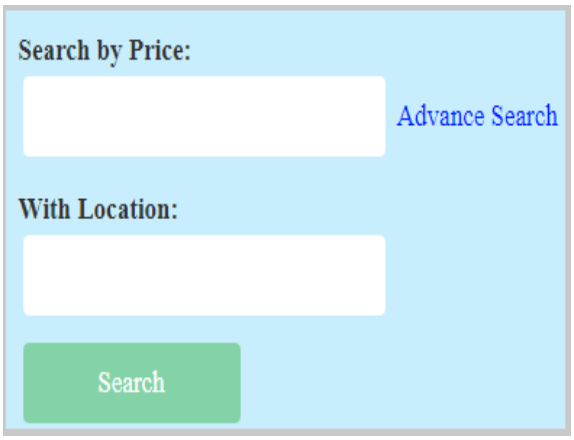

Figure 5.5: New Search Functionality

B. Structural change of web application

Structural changes refer to all the changes that modify the page layout and structure such as re-styling or changing a web element's locator. It impacts only the page layout/structure, modified to beautify the web application appearance or to reorganize its content (*e.g.*, switching from a table-based to a tableless layout). In the test suite, the test script has to modify one or more lines containing locators that are affected by the structural changes.

When the position of the web element changes on the screen, then the test scripts may fail to locate such elements via the existing locator. The algorithm solves this problem by using taking the DOM tree from the old and new versions of the applications. It then extracts the web element properties using ML, which is used for locating elements, from nodes and searches for them in the DOM tree of the new version of the web application. For all the DOM tree nodes which match the locating properties, the technique updates the locator to match these nodes and verifies if that makes the test pass. All locator updates in this phase resulting in a test case pass are returned and the successful locators are saved for the next testing purpose.

C. Presentation Change of Web Application

Dynamic web applications sometimes generate dynamic ID for web application elements. Those type of change is called presentation change of the web application. Usually, the change of DOM object attributes refers to changes in rendering or visualization options of the DOM object. It includes auto-incremented numbers, like ID=AAU_001, AAU_002. These numbers can change from session to session or from one application version to another. For example, the ID that was generated automatically was not found problem happen when presentation change occurs. This error occurs when Locator L tries to locate an element e using an automatically generated ID that worked in V but does not work in V0. The algorithm to detect an element that has this type of feature is extracting stable ID from dynamic ID.

In the example as shown Figure 5.6 the stable ID is AAU_00. For this purpose, define a regular expression pattern that denotes the meaningful part of the ID using capturing parenthesis {}: AAU_00\d+*. Depending on this outcome, the result will then automatically adjust the priority and include the ID attribute only if it makes sense.

```
<div clas="features" id="AAU_001">
the previous text
</div>
<div clas="features" id="AAU_002">
the modified text
</div>
```

Figure 5.6: DOM Presentation Change

5.5 Evaluation Criteria

The effectiveness of WELA is evaluated in terms of the number of broken locators that are repaired, locating the correct web element, and the execution time. It is done by comparing the algorithm with another related algorithm for a given DWA. i.e., how effective the proposed algorithm when the AUT or application is changed.

A. Accuracy

The accuracy of the algorithm is done by evaluating the percentage of correctly identified web elements and executed the test. The percentage is computed by taking the ratio of correctly identified web elements to the total web elements when the web application changes its structure, content, and presentation. The final result for the accuracy of the algorithm for the selected web application is calculated as shown in Table 5.3.

WELA is evaluated by asking some questions like what is the accuracy to generate the correct locator during the testing phase, how effective is it when the AUT is changed, and finally what is the performance overhead of WELA to generate the correct locator during the testing phase.

Table 5.3: Locating Accuracy of WELA for the Selected Application

Web Application	**Number of Elements per Page**	**No of Changed Web Element**	**Unable to select the Correct Web Elements after Using WELA**	**WELA Accuracy (%)**
Google	Few	Frequently Changing	No	100
Gmail	Few	Frequently Changing	No	99
Amazon	Moderate	45	15	88
Address book	Few	119	13	92
Claroline	Few	53	6	97
MantiBT	Few	49	4	97
Collabtive	Many	44	No	100
PPMA	Many	42	4	92
Industrial APP	Many	250	5	98
RBS	Few	35	No	**100**
Total			**47**	**97%**

Table 5.3 shows the number of changed web elements and the number of broken web elements when using WELA. In the experiment ten web application is present. For each web application, there are one original version and one later version. We calculate the number of broken locators for all web applications by other locator algorithms and WELA as shown in Figure 5.7. After analyzing the results in which all locators returned with the number of broken

locators, we found it very interesting that the correct probability of locating the target element in the changed web application by WELA is higher than other algorithms.

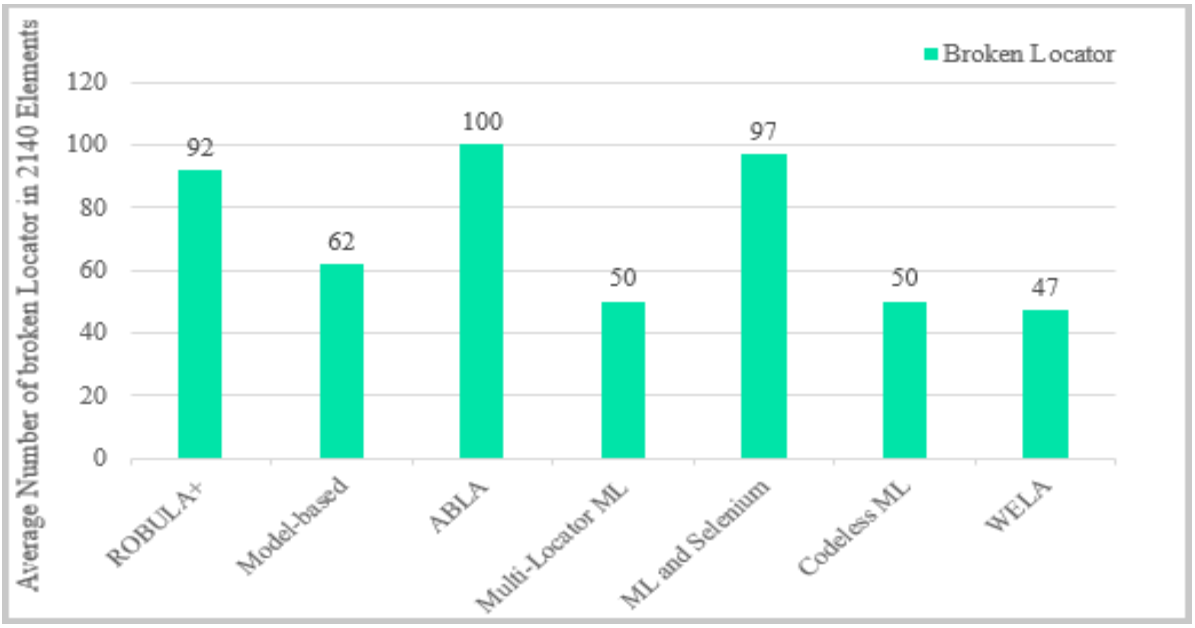

Figure 5.7: Comparison Between Algorithms for Web Element Locators

B. Execution Time

Test execution measures how long it takes to run the automated tests. Test duration is a significant metric because testing time is commonly a bottleneck in the agile development cycle [83]. With very frequent iterations to software, if tests do not run fast, teams will not run them at all. In our case, the factor affecting the total test duration of a DWA testing is the time it takes for locating the correct web element, the test time, and the execution time as shown in Figures 5.8 and 5.9. The execution time of WELA depends on the number of target elements. When the dynamic component of a web application has many input types and attributes, links, buttons, and so on, the execution time increases because the number of candidates increases, and locates each web elements required time. While locating web elements in changed web applications WELA on average took 0.8 s, this is a very short time that can be considered from the tester perspective.

Locate time=End locate time – Start time (9)

Test time= End test time – Start test time (10)

Executed time (total test duration) =End time – Start time (11)

The example in Table 5.4, shows the associated test result is displayed with the conditions on the result of the test case. Next to this, the execution time of the selected test case is also described.

Table 5.4: Sample Test Case

Test Case	**Web Application**	**Test Case Description**	**Expected Result**	**Actual Result**	**Pass/fail**
Links	Google, yahoo, Amazon, Address book, Industrial App	links and identification of items of navigation menus	Check link existence Display status	As Expected,	Pass
Sign in [Figure 5.8]	Gmail, Amazon,	Attempts to sign in using blank user name and password	Indicate error	As Expected,	Pass
		Sign in using valid username and password	Successfully sign in	As Expected,	Pass
Search [Figure 5.9]	Google, Amazon, Phonebook	Search an item and verify at least one result exist	Display search result	As Expected,	Pass
Shopping cart [Figure 5.10]	Amazon	Add product to a shopping cart,	The product added	As Expected,	Pass

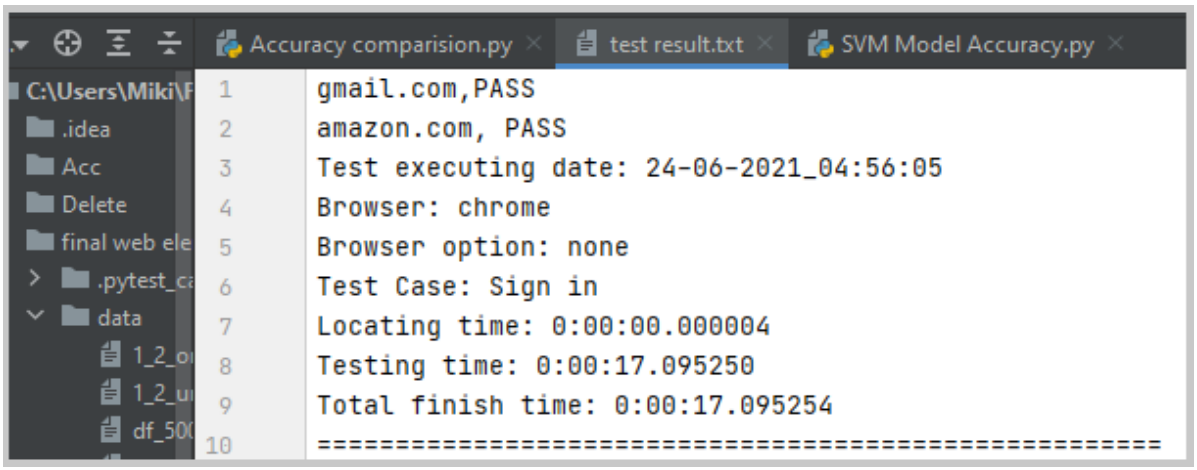

Figure 5.8: Sign in Test Execution Time

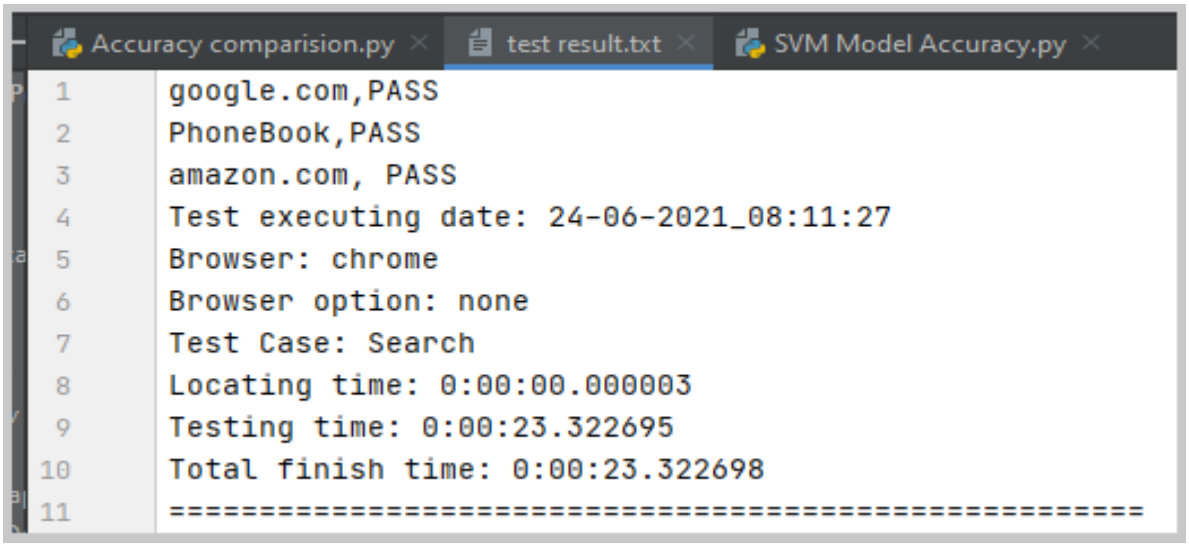

Figure 5.9: Search Test Execution Time

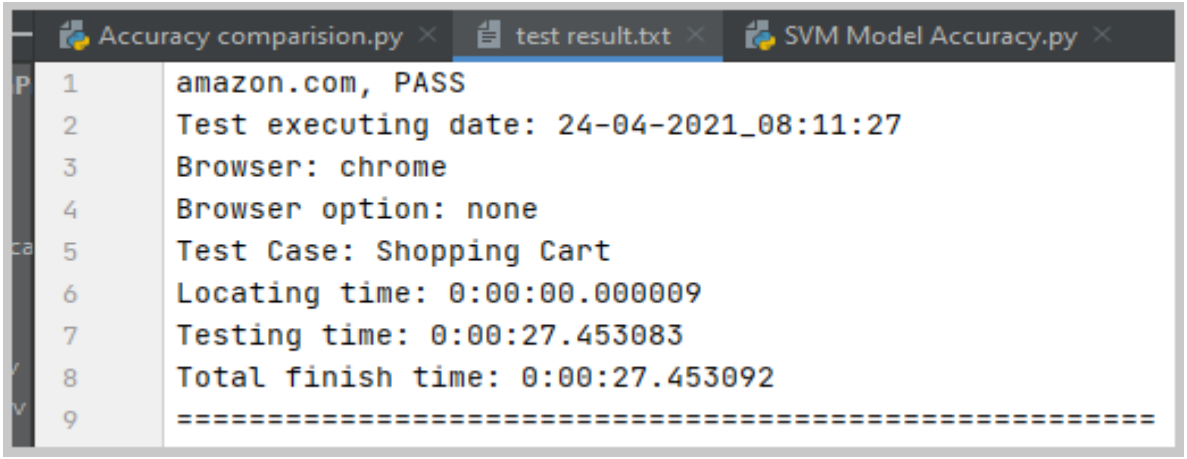

Figure 5.10: Shopping Cart Execution Time

Aldalur *et al*. [68] illustrates the execution order of an algorithms and shows the performance of each algorithm by assuming that all other factors *e.g.*, speed of the processor, are constant. Table 5.5 compares the execution time of WELA with those algorithms and other ML techniques for automatically regenerate broken locators. The time for locating a web element and automatically identify a web element with a different change of web application was calculated based on the number of tests and the number of web elements, or attributes.

Table 5.5: The Execution Order of the Web Element Locator Algorithm

Algorithm Or Techniques	**Execution Order**	**Change Consideration**	**Success (%)**	**Test Dependency**
Genetic Algorithm [17]	$O(m^n)$	Logical	87	M: test, N node attributes
ROBULA+ [21]	O(mn)	Structural	90	M: test, N Element and attributes
ABLA [68]	O(n)	Structural and Logical	100	N: nodes
Model Based [20]	$O(m^n)$	Structural, logical, and presentation	91	M: test, N Element and attributes
Multi-Locator ML [77]	O(mn)	Structural	93	M: test, N Element and
ML and Selenium. [24]	$O(m^n)$	Structural	87	M: test, N Element
Codeless ML [23]	O(mn)	Structural, Presentation	93	M: test, N Element
WELA	O(mn)	Structural, logical, and presentation	97	M: test, N Element and attributes

5.6 Discussion

This section shows the research finding and the results that have found about the problem domain with prior knowledge of the area. The proposed solution has been analyzed through experimentation to evaluate its accuracy and performance in locating web elements during DWA testing.

By automatically identifying the target web elements using linear SVM machine learning techniques, the algorithm significantly reduces test breakages caused by web element locators. We performed a set of experiments to evaluate the effectiveness of the proposed algorithm

through different scenarios. (i) when the web application changes its structure, (ii) when the web application changes its presentation), and (iii) when the web application's logical structure changes. The experiments were conducted on the selected ten open-source web applications from SourceForge, and an example of dynamic applications such as Amazon Shopping, Gmail, and Google Search. The web applications we use for the experiment change during the evolution of the web applications. As a result, existing locators become broken, resulting in locator issues such as non-selection, mis-selection, a form data problem, and an element not found. If the web application's structure did not change considerably in the successive release, locators survive the software evolution.

The effectiveness of WELA is evaluated in terms of the number of broken locators that are repaired, the accuracy of locating the correct web element, and the performance of the algorithm in terms of execution time using different scenarios. It is done by comparing the correct web element located by the algorithm with another related algorithm.

WELA was compared to studies proposed in [19], [23] [24] [21], [20] , and [68]. Earlier in Section 5.5, as shown in Figure 5.7 and Table 5.5 illustrate the accuracy and performance comparison results of the algorithm to those published techniques, respectively. Based on our findings, the suggested approach outperforms others in an average accuracy of locating the write web element and execution time.

5.7 Summary

In this chapter, an experiment is performed for testing the accuracy and performance of the developed algorithm, WELA. The experiment used Python as a development tool and Selenium as a web application testing tool. An experiment is performed over a sample of ten open-source web applications where WELA can provide a significant amount of improvement in terms of time and effort required for locating dynamic web elements during web application testing. The experiments are related to the changes of web elements that are logical change, structural change, and presentation change of a web application and address the web element locator problems. We also presented the performance and accuracy of the proposed algorithm in comparison with those previous works. The result is promising that effectively repair 97% of broken web test scripts and generate the test suites with the minimum execution time on the developed versions of a DWA.

6. Conclusion and Future Work

6.1 Conclusion

Automated testing has recently received significant attention in the software testing industry. The fundamental problem faced in dynamic web applications is that they update their contents and layouts every time. A simple modification of the AUT has an impact on locators that leads to unable to select the desired web elements on the web application because the web element that the scripts are referencing may not valid in the recent version. When web content updates, they often need to change the test scripts accordingly. Furthermore, several works have been done to identifying web elements in automated web application testing. They tried to solve the problems of test case fragility or breakage of test during automated testing by identifying web elements using different techniques in different generations.

The first generation of web element locators relied on on-screen coordinates, even if minor changes to the web application layout or screen resolution could render existing test cases useless, requiring developers to change or recreate them from scratch. For this reason, they have become obsolete and replaced by second-generation tools called DOM-based locators. Those locators overcome the limitations of the first-generation locator by accessing the web application. However, one drawback of these locators is locating web elements that do not have unique attributes because the locators generate multiple matching web elements, and it is long and flaky. Finally, the third generation of image-based web element locators, also known as visual locators, was developed. These locators used an image to find a web element of a web application during AUT.

In this research work, we have introduced an algorithm known as WELA for dynamic web application testing. The proposed algorithm covers the various changes of web elements that may cause the breakage of the web element locator by using SVM machine learning technique. The locator algorithm adapts dynamically to the change, and the aim is to save the test case breakage and effort of maintaining test scripts that spent to change or change the test codes.

Finally, in the experiments, the effectiveness of WELA is evaluated in terms of the number of broken locators that are repaired, the accuracy of locating the correct web element and performing the algorithm using different scenarios. The result is promising that effectively

repair 97% of broken web test scripts and generate the test suites with the minimum execution time on the developed versions of a DWA.

6.2 Contribution of this Work

The main contributions of this work are:

- A web element locator algorithm that improved the accuracy and performance of DWA testing is developed.
- Reusing test cases by leveraging machine learning techniques on dynamic web element locator that it is general-purpose, and reusable.
- Save time and effort in maintaining test scripts for web application testing. Failed test results prevent testers from gaining valuable insights on their tests, because the root causes may vary and do not reflect the real status and performance of the AUT.

6.3 Future Work

This research work explores the following points that can be further improved in future researches.

- These DWA testing still require human intelligence to validate and train the system and testing the selected application in the areas where ML approaches have not found solutions. As a result, developing ML capable of self-testing approaches in an adaptive environment will be effective in the future for dynamic web applications. That means fully autonomous web application testing is required, in which machines examine an application, determine what, when, where, and how testing should be conducted, and summarize the results for humans to ensure a web application is error-free and performs as expected. This aspect will need an enormous amount of data to be trained and analyzed from different perspectives.
- Locating and identifying changes of web element that occur in a logical change will involve investigating sophisticated features and considering alternative algorithms to improve the result.

References

[1] Xinyu Han, Nan Zhang, Wei He, Kai Zhang and Longli Tang, "Automated Warship Software Testing System Based on LoadRunner Automation API," in *IEEE International Conference on Software Quality, Reliability and Security Companion*, Lisbon, Portugal, July 2018.

[2] Suhaib Ahmed, Lipsa Sadath and Jumana Nagaria, "Software Testing and Lines of Codes A Study on Software Engineering Design Patterns," in *International Conference on Automation, Computational and Technology Management* , London, UK, 2019.

[3] Milad Hanna, Amal Elsayed and Mostafa-Sami, "Automated Software Testing Frameworks," *International Journal of Computer Applications (0975 – 8887)*, vol. 179 – No.46, June 2018.

[4] Rakesh Kumar, Utkalika Satapathy and Meenu Dey, "Comparative Analysis on Automated Testing of Web-based Application," in *International Conference on Advances in Computing, Communication Control and Networking*, Noida, India, 2018.

[5] Ameur Boulifa, Ana Cavalli and Stephane Maag, "Verifying Complex Software Control Systems from Test Objectives: Application to the ETCS System," in *In Proceedings of the 14th International Conference on Software Technologies*, Czech, ICSOFT 2019.

[6] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Visual vs. DOM-based Web Locators:An Empirical Study," *Casteleyn Web Engineering. ICWE*, vol. 8541, 2014.

[7] M. Hammoudi, G. Rothermel, and P. Tonella, "Why do Record/replay Tests of Web Applications Break?," in *IEEE International Conference on Software Testing, Verification, and Validation (ICST).*, Chicago, IL, USA, 2016.

[8] Ranorex, "Web Element Locators in Test Automation," 25 Sep 2019. [Online]. Available: https://www.ranorex.com/blog/web-element-locators-effectively. [Accessed 23 Jun 2020].

[9] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Reducing Web Test Cases Aging by Means of Robust XPath Locators," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, 2014.

[10] Christoph Herzog, Iraklis Kordomatis,and Wolfgang Holzinger, "Feature-based Object Identification for Web Automation," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, March 2018.

[11] Du zhang and Jeffery Tsia, "Machine Learning and Software Engineering," *Software Quality Journal*, vol. 11, pp. 87-119, 2013.

[12] D. Santiago, "A Model-Based AI-Driven Test Generation System," *FIU Electronics Theses and Dissertations,* 2018.

[13] Fumin Qi, Xiao-Yuan Jing, Xiaoke Zhu , Xiaoyuan Xie, Baowen Xu , and Shi Ying , "Software effort estimation based on open source projects: Case study of Github," *Information and Software Technology,* vol. 92, pp. 145-157, December 2017.

[14] S. Matteson, "TechRepublic," 26 Jan 2018. [Online]. Available: https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses/. [Accessed 2018 Sep 2018].

[15] I. Aldalur and O. Diaz,, "Addressing Web Locator Fragility," *in Proceedings of the ACM Symposium on Engineering Interactive Computing Systems*, vol. 17, pp. 45-50, 2019.

[16] Filipo Ricca, Maurizio Leotta, and Andrea Stoco, "Three Open Problems in the Context of Web Testing and a Vision," *Neonate Advances in Computers,* vol. 113, 2019.

[17] Amr E. Mohamed,Hadeel Mohamed Eladawy, and Sameh A. Salem, "A New Algorithm for Repairing Web-Locators using Optimization Techniques," in *13th International Conference on Computer Engineering and Systems (ICCES)*, Cairo, Egypt, 2019.

[18] Katam Reddy, Kai Petersen and Mika Mäntylä, "Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey," in *7th International Workshop on Automation of Software Test*, Zurich, Switzerland, 2018.

[19] Hiroyuki Kirinuki, Haruto Tanno, Katsuyuki Natsukawa, "Correct Locator Recommender for Broken Test Scripts Using Various Clues in Web Application," in *NTT Software Innovation Center*, Tokyo, Japan, 2019.

[20] Muhammad Iqbal, and Uzair khan, "An Automated Model-based Approach to Repair Test Suites of Evolving Web," *The journal of Systems and Software,* 2020.

[21] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "ROBULA+: An Algorithm for Generating Robust XPath Locators for Web testing," *Journal of Software: Evolution and Process*, vol. 28, pp. 177-204, 2016.

[22] Antawan Holmes and Marc Kellogg, "Automating Functional Tests Using Selenium," in *Proceedings of AGILE 2016 Conference (AGILE)*, 2016.

[23] Duyen Nguyen, and Stephae Maag, "Codeless Web Testing using Selenium and Machine Learning," in *15th International Conference on Software Technologies*, France, 2020.

[24] Nicey Paul and Robin Tommy, "An Approach of Automated Testing on Web Based Platform Using Machine Learning and Selenium," in *Proceedings of the International Conference on Inventive Research in Computing Applications (ICIRCA 2018)*, Coimbatore, India, 2018.

[25] Ken Peffers ,Tuure Tuunanen, Marcus A. Rothenberger ,and Samir Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Management Information Systems*, vol. 24, no. 3, pp. 45-78, 2007-8.

[26] Bilal Ahmad and Khaled Al Debei, "Survey of Testing Methods for Web Applications," *European International Journal of Science and Technology,* vol. Vol. 9 No. 12, 2020.

[27] Filippo Ricca and Paolo Tonella, "Analysis and Testing of Web Applications," in *Proceedings of the 23rd International Conference on Software Engineering*, Italy, 2018.

[28] A. Community, "Understand web applications," Dreamweaver, 16 May 2021. [Online]. Available: https://helpx.adobe.com/in/dreamweaver/using/web-applications.html. [Accessed 28 May 2021].

[29] Samer Al-Zain, Derar Eleyan and Yousef Hassouneh, "Comparing GUI Automation Testing Tools for Dynamic Web Applications," *Asian Journal of Computer and Information Systems (ISSN: 2321 – 5658)*, vol. 01 , no. 02, August 2013 .

[30] Shay Artzi, Adam Kiezun and, Julian Dolby, "Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit State Model Checking," *IEEE Transactions on Software Engineering*, vol. 36, pp. 474--494, 2018.

[31] Noida, and Uttar Pradesh, "Difference Between Static and Dynamic Web Pages," GeeksforGeeks, 15 Jun 2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-static-and-dynamic-web-pages. [Accessed 31 Seb 2020].

[32] A. Arora, and M.Sinha., "Web Application Testing: A Review on Techniques, Tools and State of Art," *International Journal of Scientific and Engineering Research*, no. 2, p. 3, 2017.

[33] Anuja Arora and Madhavi Sinha, "Dynamic Content Testing of Web Application Using User Session Based State Testing," in *The Next Generation Information Technology Summit (4th International Conference)*, Noida, India, 2013.

[34] R. Abbas, Z. Sultan, and S. N. Bhatti, "Comparative Study of Load Testing Tools: Apache JMeter, HP LoadRunner," *Microsoft Visual Studio (TFS), Siege. Sukkur IBA Journal of Computing and Mathematical Sciences*, p. 2, Dec 2017.

[35] Rakesh Kumar, Utkalika Satapthy and Menu Dey, "Comparative Analysis on Automated Testing of Web-based Application," in *International Conference on Advances in Computing, Communication Control and Networking*, Bhubaneswar, 2018.

[36] Tom Wissink and Carlos Amaro, "Successful Test Automation for Software Maintenance(ICSM)," in *Proceedings of the 22nd IEEE International Conference on Software Maintenance,* Philadelphia, PA, USA, 2016.

[37] T. Admin, "Explore the Scripting Techniques Used in Test Automation," Digital Thoughts , 08 Jun 2021. [Online]. Available: https://blog.thedigitalgroup.com/explore-the-scripting-techniques-used-in-test-automation. [Accessed 10 Jun 2021].

[38] J. Kent, "Test Automation From RecordPlayback to Frameworks," *Simply Software Testing Technology ,* 2019.

[39] Milad Hanna,Nahla and Mostafa , "A Review of Scripting Techniques Used in Automated Software Testing," *International Journal of Advanced Computer Science and Applications,* vol. 5, 2014.

[40] Ranjit Shewale and Sanjeev Kulkarni, "A Review of Scripting Techniques Used in Automated Software Testing," linkedin, February , 2016.

[41] K. V. Arya and H. Verma, "Keyword Driven Automated Testing Framework for Web Application," in *2018 9th International Conference on Industrial and Information Systems (ICIIS),* Gwalior, India, 2018.

[42] Uma MaheswariValli and ShanmugamValli Shanmugam, "Survey on GUI User Interface and Machine learnign based testing techniques," *Journal of Artificial intelligency,* pp. 94-112, 2014.

[43] Maurizio Leotta, Filippo Ricca, and Paolo Tonella, "Sidereal: Statistical Adaptive Generation of Robust Locators for Web Testing," *Software Testing, Verification and Reliability,* 2021.

[44] Mira Dontcheva, M. Steven. Drucker David Salesin,Michael F and Cohen, "Changes in Webpage Structure over Time," UW CSE Technical Report, Canada, 2014.

[45] Phillip Lunyov and Narges Khakpour, "Detecting Changes in Web Applications," 2020.

[46] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "Using Multi-Locators to Increase the Robustness of Web Test Cases," in *in 2015 IEEE 8th International Conference on Software Testing, Verification and Validation*, Graz, Austria, 2015.

[47] A. Knight, "Web Element Locators for Test Automation," Automation Panda, 15 Jan 2019. [Online]. Available: https://automationpanda.com/2019/01/15/web-element-locators-for-test-automation/. [Accessed 23 Feb 2020].

[48] G. Fiumara and R. Baumgartner, "Web Data Extraction Applications and Techniques," *Center for Complex Networks and Systems Research*, pp. 301-321, 2014.

[49] O. Díaz. Arellano Aldalur, H. Medina and Firmenich, "End-user Browser-side Modification of Web Pages," in *15th International Conference Web Information Systems Engineering* , Thessaloniki, Greece, 2014.

[50] Maurizio Leotta, Andrea Stoco and Filipo Ricca, "Automated migration of DOM-based Web Tests Towards the Visual," *Software Testing, Verification and Reliability,* 2018.

[51] S. Choudhry, D. Zhao and H. Versee, "Water: Web application Test Repair," in *Proceedings of the First International Workshop on End-to-End Test Script*, USA, 2014.

[52] Haenlin Michael and Kaplan Andres, "A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence," *Management Review*, vol. 61, 2019.

[53] Hussam Hourani,Ahmad Hammad and Mohammad Lafi, "The Impact of Artificial Intelligence on Software Testing," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Amman, Jordan, 2019.

[54] R. Saravanan and Pothula Sujatha, "A State of Art Techniques on Machine Learning Algorithms: A Perspective of Supervised Learning Approaches in Data Classification," in *Second International Conference on Intelligent Computing and Control Systems*, India, 2018.

[55] Suresh Sundaram, Sundararajan, Narasimhan, Savitha and Ramasamy, in *Supervised Larning with Cmplex-valued Nural Ntworks*, Berlin, Springer, 2013, pp. 125-132.

[56] X. Li and K. Wong, "Natural Computing for Unsupervised Learning," Springer International , 2019.

[57] Ankur Patel, in *Hands-On Unsupervised Learning Using Python*, Reilly Media, p. 2019.

[58] A. Nandy and M. Biswas, "Reinforcement Learning: With Open AI, TensorFlow and Keras Using Python," Apress, 2017.

[59] Faiza Khan,Summrina Kanwal, Sultan Alamri and Bushra Mumtaz, "Hyper-Parameter Optimization of Classifiers, Using an Artificial Immune Network and Its Application to Software Bug Prediction," *IEEE Access*, vol. 8, January 2020.

[60] M. C. Saputra and T. Katayama, "Code Coverage Similarity Measurement Using Machine Learning for Test Cases Minimization," in *IEEE 9th Global Conference on Consumer Electronics (GCCE)*, Kobe, Japan, 2020.

[61] Gangadhar Shobha, Shanta Rangaswamy, "Machine Learning," in *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*, 2018, pp. 197-228.

[62] Rizal Broer Bahaweres, Arif Suroso, Wahyu Hutomo, Permana Solihin, Irman Hermadi and Yandra Arkeman, "Tackling Feature Selection Problems with Genetic Algorithms in Software Defect Prediction for Optimization," in *International Conference on Informatics, Multimedia, Cyber and Information System*, Jakarta, Indonesia, 2020.

[63] Khadijah, Amazona Adorada and Kabul Kurniawan, "The Comparison of Feature Selection Methods in Software Defect Prediction," in *4th International Conference on Informatics and Computational Sciences (ICICoS)*, Semarang, Indonesia, 2020.

[64] S. Bharathraj and C. Rajesh, "Machine Learning in Test Automation," in *17th Annual International Software Testing Conference*, Pune, India, 2017.

[65] Abdul Rauf and Mohamad Alanazi, "Using Artificial Intelligence to Automatically Test GUI," in *The 9th International Conference on Computer Science & Education (ICCSE)*, Vancouver, BC, Canada, 2018.

[66] R. Moheb , M. Tarek , A. Bahgat, Abdullatif and M. Alaa, "An Automated Web Application Testing System," in *International Journal of Computer Applications*, 2019.

[67] Mouna Hammoudi, "Testing Web Applications: A Survey," ACM Digital Library, 2020.

[68] Inigo Aldalur,Felix Larrinaga, and Alain Perez, "An Algorithm for Repairing Structure-Based Locators Through Attribute Annotations," *Web Information Systems Engineering Lecture Notes in Computer Science*, vol. 12343, pp. 101-116, 2020.

[69] M. Leotta, D. Clerissi, F. Ricca, and C. Spadaro, "Comparing the maintainability of selenium WebDriver test suites employing different locators: a case study," *Proceedings of1st International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation*, pp. 53-55, 2016.

[70] Jin-lei Sun, Shi-wen Zhang and Song Huang, "Design and Application of a Sikuli Based Capture-Replay Tool," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Lisbon, Portugal, 2018.

[71] Andrea Stocco, Rahulkrishna Yandrapally, and Ali Mesbah, "Visual Web Test Repair," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, USA, 2018.

[72] Maurizio Leotta, Diego Filippo and PaoloTonella, "Approaches and Tools for Automated E2E Web Testing," *Advances in Computers*, vol. 101, pp. 193-237, 2016.

[73] Rajkumar, "How To Find Web Elements X Y Coordinates Using Selenium WebDriver," STM, 7 April 2017. [Online]. Available: https://www.softwaretestingmaterial.com/find-web-elements-x-y-coordinates-using-selenium-webdriver/. [Accessed 21 June 2019].

[74] Rahulkrishna Yandrapally, Suresh Thummalapenta , and Satish Chandra, "Robust Test Automation using Contextual Clues," in *In Proceedings of the 2014 International Symposium on Software Testing and Analysis*, New York, USA, 2014.

[75] M. Harman and N. Alshahwan, "Automated Session Data Repair for Web Application Regression Testing," in *In Software Testing, Verification, and Validation, 2018 1st International Conference*, Lillehammer, Norway, 2018.

[76] Andrea Stocco, Maurizio Leotta, Filippo Ricca and Paolo Tonella, "APOGEN: Automatic Page Object Generator for Web Testing," *Software Quality* , 2016.

[77] Yu Zheng, Song Huang, Zhan-wei Hui, and Ning Wu, "A Method of Optimizing Multi Locators Based on Machine Learning," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion*, Nanjing, China, 2018.

[78] Celia Chen, Reem Alfayez, Barry Boehm, and Lin Shi, "Why is it Important to Measure maintainability-testing , and What are the Best Ways to Do it?," in *IEEE/ACM 39th International Conference on Software Engineering Companion*, 2017.

[79] Nalindren Naicker ,Timothy Adeliyi, and Jeanette Wing, "Linear Support Vector Machines for Prediction of Student Performance in School-Based Education," *Mathematical Problems in Engineering*, 2020.

[80] A. Gholamy, "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation," scholarworks, 2018.

[81] Micael Gallego, Francisco Gortázar and Mario Munoz-Organero, "A Survey of the Selenium Ecosystem," *Electronics*, 30 June 2020.

[82] Erdinç Uzun, Tarik Yerlikaya, and Oğuz Kirat, "Comparison of Python Libraries Used for Web Data Extraction," *Fundamental Sciences and Applications*, 2018.

[83] A. Cristian, "Efficient test execution in End to End Testing : Resource Optimization in End to End Testing Through a Smart Resource Characterization and Orchestration," in *International Conference on Software Engineering: Companion Proceedings*, Korea, 2020.

Annexes

Annex A: Source Code of the Algorithm

A. Setup the Environment

Importing Dataset

```
# importing the dataset
Df_file = Path("data") / "locator_lable_Data.csv"
dataset = pd.read_csv(Df_file)
df = pd.DataFrame(dataset, columns=['name', 'id','class', 'title', 'role','aria-label', 'accesskey', 'target'])
print(df)
```

Output

```
C:\Users\Miki\PycharmProjects\algorithm\venv\Scripts\python.exe "C:/Users
                name                   id  ...         accesskey target
0      search_query               search  ...                 H     id
1             email                  NaN  ...         character  class
2                 q    footer-search-bar  ...                 C     id
3                 q         navbar-query  ...                 W     id
4             query               query2  ...               NaN     id
...             ...                  ...  ...               ...    ...
5274        as_word                 text  ...             value  class
5275            NaN              zipcode  ...               NaN  class
5276            NaN                geeks  ...  single_character  class
5277             qs             myHeader  ...               NaN  class
5278  field-keywords  twotabsearchtextbox  ...                 S     id

[5279 rows x 8 columns]

Process finished with exit code 0
```

B. Dataset Constructor (Training and testing data using the percentage split (80% Training, 20% Testing)

```
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Import svm model
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

C. CSV Data Conversion

```
# Label encoding for converting our ML categorical data to numerical data using label encoding
def Encoder(df):
    columnsToEncode = list(df.select_dtypes(include=['category', 'object']))
    le = LabelEncoder()
    for feature in columnsToEncode:
        try:
            df[feature] = le.fit_transform(df[feature])
        except:
            print('Error encoding ' + feature)
    return df
```

D. Data cleaning

```
import pandas as pd
from pathlib import Path
DF_FILE = Path('data') / "locator_lable_Data.csv"
empty_keys = ['name', 'id', 'class', 'title', 'role', 'aria-label', 'accesskey', 'target']
df = pd.read_csv(DF_FILE)
# Check if attributes have empty value
df2 = df[df[empty_keys].isnull().all(axis=1)]
print(df2)
```

Output

```
C:\Users\Miki\PycharmProjects\algorithm\venv\Scripts\python.exe
     name   id class title role aria-label accesskey target
88    NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
189   NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
255   NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
371   NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
679   NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
...   ...  ...   ...   ...  ...        ...       ...    ...
5201  NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
5205  NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
5228  NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
5237  NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN
5254  NaN  NaN   NaN   NaN  NaN        NaN       NaN    NaN

[65 rows x 8 columns]

Process finished with exit code 0
```

D. **Selecting the Write Web Element Locator**

```
RAW_FILE = Path("data") / "1_2_urldata.csv"
DF_FILE = Path("data") / "locator_lable_Data.csv"
element_key = ['name', 'id', 'class', 'title', 'role', 'aria-label', 'accesskey', 'target' ]
def label_data(dict):
    if dict['name']:
        dict.update({'target': 'name'})
    elif dict['id']:
        dict.update({'target': 'id'})
    elif dict['class']:
        dict.update({'target': 'class'})
    elif dict['title']:
        dict.update({'target': 'title'})
    elif dict['aria-label']:
        dict.update({'target': 'aria-label'})
    elif dict['accesskey']:
        dict.update({'target': 'accesskey'})
d2 = {}  # A list to store the clean data
clean_data = []
with open(RAW_FILE, 'r', encoding='utf-8') as rf, open(DF_FILE, 'w', encoding='utf-8', newline='') as dfFile:
    reader = csv.reader(rf, delimiter=',')   # open file, read raw data from each row then process data:
    for row in reader:
        # convert raw data to dictionary
        element_dict = {k: v.strip('"') for k, v in re.findall(r'(\S+)=(".*?"|\S+)', row[1])}
        # add only interested attributes
        d2 = {k: element_dict.get(k) for k in element_key}
        label_data(d2)
        clean_data.append(d2)
```

F. **Compiling the Algorithm WELA**

```
# Feature columns
X = df.drop(['name', 'target'], axis=1)
y = df['target']
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Create a svm Classifier
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
# Predict the response for test dataset
y_pred = svclassifier.predict(X_test)
print("Confusion matrix")
print(confusion_matrix(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such
print("Classification Report")
print(classification_report(y_test, y_pred))
# Model Accuracy
print("The accuracy of the Model is ", accuracy_score(y_test, y_pred))
```

```
C:\Users\Miki\PycharmProjects\algorithm\venv\Scripts\python.exe
Confusion matrix
[[878   0 ]
 [  0  149]]
Classification Report
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       900
           1       0.87      0.97      0.92       154

    accuracy                           0.97      1054
   macro avg       0.93      0.97      0.95      1054
weighted avg       0.98      0.97      0.97      1054

The accuracy of the Model is  0.9743833017077799

Process finished with exit code 0
```

Model	Precision	Recall	F1-Score	Support	Accuracy (%)
Linear SVM	0.98	0.97	0.97	1054 (25%)	0.97
Percentage Split (80)					

Annex B: Features of Web Element

Features	**Description**
Name	Define the Name of form web element
Class length	Length of the class attribute of the web element
Parent HTML Tag	The tag for the given element's parent.
Num. Children	The number of HTML nodes that are children of the given element's node.
Num. Siblings	The number of HTML nodes that are siblings of the given element's node.
Attributes	Each attribute of the element
Vertical Percent	The relative vertical position (in percentage) of the given element.
ID	Unique identifier of the DOM node

Distance from Input	The relative (normalized against the full set of elements) distance to the closest input widget from the given element.
Type	The string to be set to an input field.
Text	The actual text is associated with the given element.
Relative width, relative height, relative x, relative y	Position and dimension of the element related to the enclosing document's position

1	Tag_Name	Type	Id	Name	class	value	web element
2	input	text	username	username	loginname	no	Text Area
3	input	text	combo-1	type	x-form	no	dropdownmenu
4	input	text	combo-1	type	x-form	no	dropdownmenu
5	input	text	combo-1	type	x-form	no	dropdownmenu
6	input	text	uname	uname	loginname	no	Text Area
7	input	passwoer	pass	pass	loginpass	no	Text Area
8	input	passwoer	password	password	loginpass	no	Text Area
9	input	submit	NA	submit	loginbt	login	Button
10	select	option	cars	cars	NA	Volvo	dropdownmenu
11	button	submit	no	no	btn-primary		Button
12	select	option	cars	cars	no	Merceds	dropdownmenu
13	input	radio	no	no	no	no	radiobutton
14	input	checkbox	no	chk[]	checkbox-	kichen	checkbox
15	input	checkbox	no	chk[]	checkbox-	Laundry	checkbox
16	button	submit	ID-3465-text1		ID-Class-text45		Button
17	button	submit	ID-4434-textE2		ID-Class-text73		Button
18	input	text	no	q	gLFyf gsfi	search	Text Area

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared By:

Name: Mikiyas Bayew Bekele

Signature: _____

Date: _____

Confirmed by advisor:

Name: Mesfin Kifle (PhD)

Signature: _____

Date: _____

Place and date of submission: Addis Ababa August, 2021