AN AUTOMATED TESTING TOOL BASED ON GRAPHICAL USER INTERFACE WITH EXPLORATORY BEHAVIOURAL ANALYSIS

MS.K.JAGANESHWARI 1, DR.S.DJODILATCHOUMY 2

1Research Scholar, SCSVMV University, Kanchipuram.
Assistant Professor, CTTE College, Chennai, India.
2Head of the Department of Computer Science, Chellammal Women's College, Chennai, India.
Email: 1kjaganeshwari@gmail.com, 2djodilatchoumy@hotmail.com

ABSTRACT

Web-based applications have complex mechanisms, and it is challenging to perform any test. Computerization examination applies automation tools to decrease individual interference and repeatable assignments. In this article, we have designed and implemented an automation testing framework for web applications. The Selenium WebDriver tool was used to execute this innovative automated testing model. With this structure, testers can match the widgets with pre-trained dataset by taking screenshots of the web pages and by clicking automatically using PyAutoGUI. The screenshot property of the framework is useful to creators for evaluating their design of the web page. We introduced a novel methodology for widget detection, widget classification and testing coverage using machine learning and image processing concept. In this approach, we can give the URL as input, so, the GUI widget images can be quickly captured from the server and do not require large storage repositories to be used as training samples. Widgets of GUI images are detected from the pre-trained datasets by applying BLOB text detection. After identifying the widgets, they are classified into domain-specific categories, for example, labels, buttons, input boxes, check boxes, and links). Finally, it is evaluated to achieve a mean GUI component categorization of superior precision. Previous works have mostly used Java GUIs, Python GUIs, and Visual Basic GUIs to distinguish the types of widgets, but in our work, we observed the types of widgets using Image Processing. From this automated testing implementation on web applications, we have achieved 98.4% of test coverage accuracy.

Keywords: Automated Testing, GUI, DNN, Computer Vision, Image processing

1. INTRODUCTION

Graphical User Interfaces (GUIs) are significant component in most of the current software. In addition to extensive testing of the GUI, the fundamental code also requires undergoing massive testing regime to make certain that it performs accurately [1]. Also, In the graphics user interface (GUI) mapping, many functions in the image processing (IP) field are integrated into a call back so that it can perform actions related to IP functions, including image detection and object classification [2]. A secondary image can be created based on the provided functionalities after loading any of the image file types.

For an appropriate functioning of the web-based applications requires software systems. So, the web applications quality is a main factor during deployment of these applications. To enhance the quality of software, repeated testing plays a crucial position [3]. As a result of automation testing, software quality is improved, and human intervention is minimized. The testing tasks can be accomplished with a variety of open-source and commercial tools, such as Watir, JMeter, Selenium, and QTP. The Selenium automation tool is commonly used open-source kits for testing the web applications [4]. The main idea of this paper is to develop automated testing tool with computer vision techniques. Also, this research involves on image processing techniques such as feature extraction, widget detection, widget classification, testing coverage & comparison.
2. RELATED WORK

In [1], discovered a new way to improve GUI testing by applying machine learning techniques to recognize and classify GUI widgets. Moreover, we found that using training sample, URL linkages, and screen links, we could recognize and categorize GUI items in screenshots and report depending on their positions (x, y coordinates) and types. For improving the GUI testing by utilizing machine learning techniques to automatically detect GUI widgets in screen photos [2]. We produce randomized GUI as trained data in order to extract widget information automatically. We contemporary a neural device decipherer that utilizes the modern developments in computer image and instrument interpretation to translate UI design images into GUI skeletons [3]. We employ mature approaches from the computer vision (CV) region to recognize GUI elements, including old-school method that depend on classic image processing characteristic and deep learning techniques trained to detect massive volumes of GUI data [4]. Nevertheless, these CV techniques were not designed with the unique characteristics of GUIs and GUI components in mind [2], or with the superior localization precision required for the GUI elements recognition assignment. Simple and efficient image annotation model use Convolutional Neural Network (DNN) extracted features from a pictures and word embed vectors to represent their relevant tags [12]. The Canonical Correlation Analysis (CCA) framework, which aids in modelling both text and visual components of the dataset, is the foundation for our first set of models. UIED (User Interface Element Detection), a toolkit developed to give users a basic and easy-to-use platform for detecting GUI elements accurately [6]. To grip various and intricate GUI pictures, UIED includes multiple detection algorithms, including traditional computer vision (CV) methodologies and deep learning models [2]. UIED has been found to be capable of accurate detection and beneficial in subsequent projects. The depth of the analysis is decided about how many of the program capabilities are covered [13]. Typically, these solutions include executing an app in a sandbox. For enormous number of applications and limits human resources, auditor find it difficult to manually execute the app. Mobile app inputs are often represented via interactions with the app's graphical user interface (GUI).

As the authors have established in modern software development [7], system testing is used to validate requirements conformance at all stages of system abstractions, including the system's graphical user interface. GUI-based automated tests, like all other automations, aim to cut testing costs and time in half when compared to human procedures [5]. In industrial practise, automated testing has proven to be efficient in lowering costs for a variety of tests (such as unit or integrated testing). For covering all conceivable GUI pathways with test scripts would be extreme time consuming and result in major maintenance concerns [8]. We propose using the open-source TESTAR tool to supplements scripted testing with script less test automation. Jonathan A. Saddler produced Event Flow Slicer. A GUI tester can use this tool to develop and create all of the actual test cases needed to accomplish a certain goal. The user registers pertinent actions [11] on the UI before setting limitations to narrow the scope of the activity. To acquire only the widgets that are relevant to that aim, an events flow graph is used. The author created a trained network that can predict widget location and size from display images, which has an impact on where and how GUI testers interact with the software [10]. Our SM gives an overview of existing GUI testing methodologies and assists in identifying areas in the industry that deserve greater research attention. Connecting theoretical model-based methodologies with commercially available technologies, for example, will take a lot of effort [9], to that objective, investigations comparing the hi-tech in GUI tested in academic methodologies and modern tools are required. Deep learning is a framework that uses cutting-edge deep icon-behaviour learning to train and detect intention-behaviour mismatches in a variety of popular apps [14]. Deep learning, in particularly, use programs analysis approaches to relate UI widget programme behaviour to their intentions and inferred UI widget labels from the programmer behaviours, enabling for the production of a large-scale, high-quality training sample. First, there is a gap between the programmers' purpose and the outputs writing query, as well as a gap between the textual question and the visual GUI designs. Secondly, other developer may use the same GUI designs, resulting in high levels of similarities to other apps and lowering the application's uniqueness. Finally, several of the retrieved GUI may have outdated design style, making it difficult for programmers to keep up with current GUI design trends [15].

3. PROPOSED APPROACH

The main objective of this research is to present a data-driven approach for automatic prototype testing software for web applications and its...
implementation. And to explore and analyse the adoption of computer vision approaches in the field of software testing. To detect different GUI widgets, this design employs the testing concept assessment archives in the Python OpenCV library [1]. In addition, the novelty of this proposal is that we are providing the URL an input, so it immediately captured from the server and does not demand enormous sources of the GUI widget illustrations as training experiments. As an alternative, the GUIs are detected only on its leaping boxes and color deviation. The expert ideal dataset includes input data and reaction assessments. This development unsurprisingly generates precise tasks such as recognition, categorization, and assessment analysis. Profoundly, widgets of GUI images are associated from the unsubstantiated datasets using computer visualization systems. Later, individuals classified widgets are organized into domain-specific categories (eg. labels, buttons, input box, check box, and links, etc.) [1]. Ultimately, it is calculated to attain a mean GUI component categorization for superior precision. Recent literatures have used Java GUI, Python GUI, and VB GUI for detecting the type of widgets, but we have identified the type of widgets using Image processing methods [1]. Image processing based on machine learning that simulates human vision and automates image analysis [2].

3.1. Generation of Screenshot:

Selenium web-driver does not encourage the screenshot for pass and malfunction test cases [6]. We have applied novel work that will get the screenshot for pass and malfunction situation [7]. Utilizing this type of function tester can simply pick up the in accuracies transpired in web application. This method will aid the designer to investigate their testing. After completing the test suite, screenshots of success and failure widget links are stored in catalogue according to network smart database. Measures to create screenshot: 1. Build catalogue where you store the screenshot of pass and failure widgets and links [8]. 2. Apprehend the result from selenium web driver. Validate the result is pass or fail 4. If the result is pass or fail, the screenshot for web page is obtained. 5. Place the network name as the filename for screenshot image. 6. Store up the image file in distinct index [8].

4.ENTITYSOURCE

Selenium web-driver endorses various forms of sensor to uncover the web page components [9]. To operate the selenium web driver, we have to schedule languages to imprint automation screenplays style [10]. Essentially, no matter what the language that we primarily chosen should have a selenium client library friendly [11]. All the selenium web driver highlights deliver gain access to the selenium web driver client collection for python. Broadly, we ought to recognize that the selenium impartial server for isolated and scattered testing for browser-based functions [12].

Web page features can be situated by labels, text boxes, combo boxes, buttons etc. Entity source store up all the widgets of web page sections. This will streamline the assignment of sustaining and refurbishing the practice circumstances. For example, previous version of web application contains ‘Login’ button. Subsequent type of web application ‘Login’ button changed to ‘Login Now’, so it is required to train the widgets [13]. To prevent any challenges, we have implemented object repository which contains the trained samples [14]. Maintenance cost will be reduced as well as the storage space. At any time, transformation arises in web application elements, tester no needs to change the object repository [14]. Unlike the existing architecture, it does not require huge repositories as training samples [1]. Here we have used the Frozen East Text detector dataset. OpenCV’s EAST text detector dataset is DNN prototype, built on a new structural design and guidance configuration. It is adept of operating at close instantaneous at 13 FPS on 720p images and it acquires hi-tech text recognition precision. And it uses OpenCV’s EAST sensor that’s spontaneously distinguish text in both images and video streams [15].
5. THE PROPOSED METHODOLOGIES-WIDGET DETECTION AND WIDGET CLASSIFICATION

This work tries to exploit the unique approach of investigative developmental detection. We have used the PyAutoGUI automation python library for controlling the mouse to click the numerous networks in the certain n URL [1]. By way of scrutinizing the subsequent screenshot using the OpenCV object recognition method for new discrepancies and GUI classification can be done by beyond engaging a DNN classifier. If the subsequent screenshot is solely new than the original GUI image, then the GUI can be recognized as a Hyperlink/ Button [1]. To implement this, we have defined the function called image decision function. The algorithm is given below:

Algorithm for Image Decision Function

Step 1: import cv2, numpy, imutils
Step 2: Define Image decision function with parameters name_1, name_2 and verbose
Step 3: Read source image and target image using cv2.imread
Step 4: Convert Color Image to Gray Image using cv2.cvtColor
Step 5: Resize the Second grey image using cv2.resize
Step 6: Perform Gaussian Blur to the Source Grey Image and Target Grey image using cv2.GaussianBlur
Step 7: Apply absolute difference for the first grey and second grey images using cv2.absdiff
Step 8: Obtain threshold value to a variable threshold using cv2.threshold
Step 9: if (threshold == 1)
  Step 9.1: obtain minimum value of rows and column
  Step 9.2: Obtain maximum value of rows and column
  Step 9.3: Calculate the height = max(rows) and width=max(column)
  Step 9.4: Display diff and threshold values
  Step 9.5: Assign orig_width=height and orig_height=width
  Step 9.6: resv=orig_res=calc_res
  Step 9.7: diff_percent = (calc_res/resv/resv)*100
  Step 9.8: if (diff_percent > 70) then img_decision = Button/Anchor Link
  Step 9.9: else if(diff_percent > 50) then img_decision = Combo/List Box
  Step 9.10: else if(diff_percent > 20) then img_decision = Input Box
  Step 9.11: else return image decision as Label
  Step 9.12: Assign np_array to args to and load the input image and grab the image dimensions using cv2.imread
  Step 9.13: Make a copy of the image, assign to args and get the shape of the image and assign it to (BW)
  Step 9.14: Set the new width and height and then determine the ratio in change for both the width and height using (W=width/orig_width) and (H=height/orig_height)
  Step 9.15: Resize the image and assign it as image and get the array options
  Step 9.16: Assign the cv2.imshow function to cv2.addWeighted('Name_1', 'Name_2', 'Name_3', 0.5, 0) to not stop
  Step 9.17: Construct a block from the image and then perform a forward pass of block using cv2.addWeightedImage
  Step 9.18: Assign the function using not forwarded to numpy
  Step 9.19: Print the Object decision time taken in seconds using the standard format specified
  Step 9.20: Using with image from 0 to numFiles

5.1. Widget Detection

This work used Open-Source Computer Vision Library (OpenCV). It is an icon managing tool with an open-source computer vision concept and object knowledge software. Astonishingly, it is the swiftest prototyping of computer image challenges. OpenCV python zips to the trepidation of videos, install images, and locate objects with both a normal webcam and a focused depth sensor. It carries refined boundaries for capturing, processing, and conferring picture data. OpenCV’s Python supplements can assist us reconnoiter vindications to these requirements in a high-level language and in a homogeneous data format that is interoperable with scientific libraries such as NumPy, beautifulsoup, imutils [16]. Although construing info from replicas vintages extremely precise leapfrog containers for GUI components, this data is not forever accessible due to restrictions in the photo-editing software used or alterations in design practices, such as delineating mock-ups digitally or physically using pen demonstrations, tablets, or paper. A mock-up relic could be nothing more than an image in some cases, necessitating the use of CV techniques to distinguish key GUI-component information. To accommodate these circumstances, our solution employs CV systems to recognise GUI component bounding boxes. To inferred joining frames across items comparable to GUI mechanisms in a picture, this procedure retains a variety of CV techniques. The edges of objects in an image are first detected using BLOB text detection algorithm. The edges are then diluted to blend edges that are close together. Finally, leapfrogging frames over atomic GUI-modules are calculated using the contours of those edges. The algorithm is given below.

Algorithm for BLOB Text Detection

Step 1: import cv2, numpy, imutils, time, argparse, scikit-image, Venn
Step 2: Define handler function and Assign the Options for Choose options using add arguments
Step 3: Assign the webdriver.chrome to the drive and place the Test for URL input.
Step 4: To find the element with longest height on page and save the driver screenshot using save_screenshot and quit driver
Step 5: Read the screenshot using cv2.imread, assign to img and show the image using cv2.imshow
Step 6: Assign the value of ArgumentParser to np
Step 7: Determine the ArgumentParser with add_argument with type=int, ‘i’, ‘image’ and help=“Path to input image”
Step 8: Determine the ArgumentParser with add_argument with type=int, ‘i’, ‘text’, and help=“Path to input EAST text detector”
Step 9: Determine the ArgumentParser with add_argument with type=int, ‘i’, ‘text’, and help=“Path to input EAST text detector”
Step 10: Determine the ArgumentParser with add_argument with type=int, ‘i’, ‘text’, and help=“Minimum probability required to imprint a region”
Step 11: Determine the ArgumentParser with add_argument with type=int, ‘i’, ‘text’, and help=“Maximum value width should be multiple of 32”
Step 12: Assign np array to args to and load the input image and grab the image dimensions using cv2.imread
Step 13: Make a copy of the image, assign to args and get the shape of the image and assign it to (BW)
Step 14: Set the new width and height and then determine the ratio in change for both the width and height using (W=width/orig_width) and (H=height/orig_height)
Step 15: Resize the image and assign it as image and get the array options
Step 16: Assign the cv2.imshow function to cv2.addWeighted('Name_1', 'Name_2', 'Name_3', 0.5, 0) to not stop
Step 17: Construct a block from the image and then perform a forward pass of block using cv2.addWeightedImage
Step 18: Assign the function using not forwarded to numpy
Step 19: Print the Object decision time taken in seconds using the standard format specified
Step 20: Using with image from 0 to numFiles

i) Before WidgetDetection- Web page of yahoo.com captured as screenshot before detecting the widgets using image processing techniques python openCV.
Figure 4: screenshot of yahoo.com before widget detection

ii) After Widget Detection- Web page of yahoo.com captured as screenshot after detecting the widgets using image processing techniques using python openCV. And the widgets are highlighted in green rectangular box using bounding boxes techniques.

Figure 5: Widget detection of Yahoo web page.

The widget detection time of various real time websites are tabulated and given below:

<table>
<thead>
<tr>
<th>URL of Web Pages</th>
<th>Object detection time</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.gmail.com/">https://www.gmail.com/</a></td>
<td>9.958827 seconds</td>
</tr>
<tr>
<td><a href="https://www.google.com/">https://www.google.com/</a></td>
<td>9.760312 seconds</td>
</tr>
<tr>
<td><a href="https://services.gst.gov.in/services/login">https://services.gst.gov.in/services/login</a></td>
<td>10.047040 seconds</td>
</tr>
<tr>
<td><a href="https://www.instagram.com/accounts/login/">https://www.instagram.com/accounts/login/</a></td>
<td>9.672259 seconds</td>
</tr>
<tr>
<td><a href="https://www.linkedin.com/login">https://www.linkedin.com/login</a></td>
<td>9.711001 seconds</td>
</tr>
<tr>
<td><a href="https://profile.w3schools.com/log-in">https://profile.w3schools.com/log-in</a></td>
<td>9.824069 seconds</td>
</tr>
<tr>
<td><a href="https://login.skype.com/">https://login.skype.com/</a></td>
<td>9.810490 seconds</td>
</tr>
<tr>
<td><a href="https://www.snapdeal.com/login">https://www.snapdeal.com/login</a></td>
<td>10.351611 seconds</td>
</tr>
<tr>
<td><a href="https://www.webiste.com/sign-in/">https://www.webiste.com/sign-in/</a></td>
<td>10.276383 seconds</td>
</tr>
<tr>
<td><a href="https://login.yahoo.com/">https://login.yahoo.com/</a></td>
<td>9.743537 seconds</td>
</tr>
<tr>
<td><a href="https://www.amazon.in/sp/signin">https://www.amazon.in/sp/signin</a></td>
<td>9.633661 seconds</td>
</tr>
</tbody>
</table>

Table 1: Object detection time of web pages

The widget detection time of various real time websites are represented in graphical format and given below:

Figure 6: Widget detection time of yahoo.com in the Python IDE
5.2. Widget Classification

It recommends a unique approach of investigative developmental recognition. We have applied the PyAutoGUI automation python library for controlling the mouse to click the several connections in the particular URL [1]. Following the pick-up of the resultant screenshot employing the OpenCV object recognition procedure for new modifications and GUI classification, which can be done by additional commissioning a DNN classifier [18]. If the resultant screenshot is totally different than the prototype GUI imaged, then the GUI can be classified as a Hyperlink/ Button. If the Cursor image is activated, it can be discovered as an Input box or if within the same page a list of items is shown, it can be recognized as a Combo Box [19]. This assessment approach demands to be trialled with different real-time websites for additional authenticity and precision. Accordingly, without controlled training samples, the suggested approach can generate precision in GUI component recognition and categorization. Blob text detection algorithm use computer vision to detect regions of a digital image that differ from surrounding regions in terms of properties, such as brightness or color [20]. This algorithm is needed to classify the extracted text blobs into horizontal text regions. This process is also referred to as text segmentation [21]. This classification is based on the size and position of the blobs. Image classification algorithm with exploratory behavioural analysis algorithm is used to classify the widgets. The algorithm is given below:

![Figure8: Image classification with exploratory behavioural analysis algorithm](image)

The report with x,y coordinates of widgets positions, matched percentage of widgets with trained samples, type of the widgets, URL Link, and the screen link (.png)of Google.com is downloaded as .csv file, which is given below:
The widget classification of Google.com website is represented as graphical format with the position of the x, y co-ordinates of widgets and given it is displayed below:

load googlet.txt

plot(googlet,'DisplayName','googlet');
xlabel("x values"); ylabel("y values");

The widget classification of Google.com website is represented as graphical format with percentage of matched widgets with trained data sample and given it is displayed below:

xlabel("widgets"); ylabel("%matchedwidgets");

6. TEST COMPARISON

GUI widget detection, classification accuracy, GUI categorized link accuracy, testing coverage probability are analysed with many real-time scenarios and which is tabulated and displayed below:

<table>
<thead>
<tr>
<th>Websites</th>
<th>Existing Categorized Links</th>
<th>Test coverage in %</th>
<th>Proposed Categorized Links</th>
<th>Test coverage in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google.com</td>
<td>30</td>
<td>98</td>
<td>31</td>
<td>99</td>
</tr>
<tr>
<td>Twitter.com</td>
<td>24</td>
<td>97</td>
<td>25</td>
<td>98</td>
</tr>
<tr>
<td>Gmai.com</td>
<td>20</td>
<td>98</td>
<td>21</td>
<td>99</td>
</tr>
<tr>
<td>Yahoo.com</td>
<td>24</td>
<td>96</td>
<td>26</td>
<td>98</td>
</tr>
<tr>
<td>Skype.com</td>
<td>18</td>
<td>95</td>
<td>21</td>
<td>99</td>
</tr>
<tr>
<td>Snapdeal.com</td>
<td>32</td>
<td>94</td>
<td>34</td>
<td>98</td>
</tr>
<tr>
<td>LinkedIn.com</td>
<td>33</td>
<td>97</td>
<td>28</td>
<td>99</td>
</tr>
<tr>
<td>Instagram.com</td>
<td>33</td>
<td>97</td>
<td>28</td>
<td>99</td>
</tr>
<tr>
<td>Amazon.in</td>
<td>23</td>
<td>96</td>
<td>25</td>
<td>98</td>
</tr>
<tr>
<td>W3school.com</td>
<td>33</td>
<td>97</td>
<td>36</td>
<td>99</td>
</tr>
</tbody>
</table>

Table 2: Test coverage of Existing and Proposed system

7. RESULT AND DISCUSSION

In the existing work of literature, a deep learning method of prototyping a GUI with supervised training is done [1]. A labelled image repository is used as training samples and DNN architecture is used to classify the GUI components [1]. Finally, it is used to derive a GUI code based on a mock input image. This idea spikes many curious possibilities in software GUI testing [1]. Due to this problem, novel methodologies are developed to find out the accuracy in automated testing. The Proposed method
is compared with existing system for test coverage accuracy. And the proposed method captured links are compared and categorized with existing system categorized links. The accuracy of the proposed system is compared with existing system using Root mean square error (RMSE), Mean square root (MSE).

The RMSE can plot a difference between the predicted value and observed value of a parameter of the model and we can find the efficiency of the testing model. So, the RMSE lower value shows the better results. And our results found that the proposed testing model has lower RMSE value compared to existing testing model.

Root Mean square (RMSE) = \[ \sqrt{\frac{\Sigma (Pi - Oi)^2}{n}} \]

<table>
<thead>
<tr>
<th>Mean Squared Error(MSE)</th>
<th>Existing</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Value</td>
<td>Observed Value</td>
<td>MSE</td>
</tr>
<tr>
<td>100</td>
<td>98</td>
<td>0.4</td>
</tr>
<tr>
<td>100</td>
<td>97</td>
<td>0.5</td>
</tr>
<tr>
<td>100</td>
<td>98</td>
<td>2.285714</td>
</tr>
<tr>
<td>100</td>
<td>95</td>
<td>4.166667</td>
</tr>
<tr>
<td>100</td>
<td>94</td>
<td>7.2</td>
</tr>
<tr>
<td>100</td>
<td>95</td>
<td>6.25</td>
</tr>
<tr>
<td>100</td>
<td>97</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>96</td>
<td>8</td>
</tr>
<tr>
<td>100</td>
<td>97</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 4: MSE of Existing and Proposed system

The Mean squared error (MSE) measures the amount of error in statistical model. It finds the average squared difference between the observed and predicted values. When a testing model has no error, then the MSE is equal to zero. Our proposed testing model values are close to zero.

Mean Squared Error (MSE): = \[ \sqrt{\frac{\Sigma (Pi - Oi)^2}{n}} \]

Table 5: Categorization Accuracy of Existing and Proposed system

The below table shows the parameters like overall pass rate, overall failure rate, execution time and maintenance cost of proposed system and existing system [23]:

<table>
<thead>
<tr>
<th>Existing system Categorization Accuracy in %</th>
<th>Proposed system Categorization Accuracy in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>98.3</td>
<td>98.4</td>
</tr>
</tbody>
</table>

Table 5: Categorization Accuracy of Existing and Proposed system

Finally testing coverage and categorization accuracy of existing and proposed system is tabulated and presented below:
8. CONCLUSION

In this article, we have suggested innovative computerization examination structure to assess the web-based testing function centred on selenium web-driver. With the intention of assessing the web application anticipated automation structure certainly diminishes the moment necessary to compose the test cases and expand the pass percentage of test coverage. It also diminishes frenetic capability of a tester. By applying this structure, one can create the tailored test statements and also investigate the malfunctions using screenshots of failed test cases. Tester can maintain all data from central place. This framework is very effective for enthusiastically altering web applications. Like this, automation framework helps businesses to test web applications economically. Also, this research is concentrated on automation testing of real-time web applications of the User Interface and User Experience. This instrumental functions economically in a reduced amount of time with fewer maintenance cost. This framework produces the tailored test report for testers. It is convenient, streamlined, and superior compared to other works available on the current literature. Thus, without supervised training samples the proposed method can yield accuracy in GUI component detection and Classification [1]. The main contribution is to reduce the investment of human resources, training and testing time. Also, this work focused on automation testing of real time websites of User Interface and User Experience. The limitations of the work are, we have used single dataset and we didn’t find the 100% accuracy in this work. So, the future enhancement of this work is a testing model with more datasets, accuracy and involves efficient statistical measures with large number of web links will be achieved.

REFERENCES


