

VOICE CONTROLLED ACCESSIBILITY AND TESTING TOOL
(VCAT)

by
Nagendra Prasad Kasaghatta Ramachandra

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science at
The University of Texas at Arlington
August, 2019

Supervising Committee:

Christoph Csallner, Supervising Professor

Elizabeth Diaz,

John Howard Robb

Copyright © by Nagendra Prasad Kasaghatta Ramachandra

All Rights Reserved

2019



DEDICATION

I dedicate my thesis work to my loving parents K.A. Ramachandra Murthy and N. Kathyayini. I am grateful for their boundless sacrifices, devotion and words of encouragement that pushed me to achieve my dreams.

My sisters K.R Malini and Sarika Shreya has been very special for being on my side at all times. All 4 of you have been my best cheerers.

I would like to thank Jack Wayne Irvin, Jr for supporting me with valuable feedback and suggestions on the tool and helped it take its current shape.

I also dedicate this thesis to my family and friends for believing in me and supporting me emotionally when I needed the most. Their selfless care and well wishes will always be appreciated.

August 2, 2019

ACKNOWLEDGMENT

I would like to offer my sincere appreciation to Dr. Christoph Csallner for his guidance and support throughout my graduate academic career. Dr. Christoph Csallner has been a wonderful mentor to me, providing invaluable technical and academic guidance over the course of this research. His knowledge and advice has always provided me with great inspiration and motivation in dealing with academic and personal challenges. Without his guidance and support, I wouldn't have reached my goal.

I thank Dr. Elizabeth Diaz and Dr. John Howard Robb for their roles on my thesis committee. I am inspired so much by their unstoppable passion and hard work for educating. None of this would be possible without these educators.

I am grateful for the love and support of my family. The completion of this thesis will mean a lot to them. I am fortunate enough to be part of them.

August 2, 2019

ABSTRACT

Voice-Controlled Accessibility and Testing tool (VCAT)

Nagendra Prasad Kasaghatta Ramachandra, MS

The University of Texas at Arlington, 2019

Supervising Professor: Christoph Csallner

Most current browser-based web applications and software engineering tools, such as test generators and management tools, are not accessible to users who cannot use a traditional input device, such as a mouse and/or a keyboard. To address this shortcoming, this research leverages recent speech-recognition advances to create a chrome browser extension that interprets voice inputs as web browser commands and executes those commands within the browser. As a result, the Voice Controlled Accessibility and Testing tool (VCAT) leverages the Chrome browser to achieve higher accessibility, with the capability to perform webpage navigation using voice commands. The tool is also capable of programmatically generating Java Selenium test case source code from the given set of voice commands, which helps achieve faster test case generation, compared to traditional methods of writing Java Selenium source code.

TABLE OF CONTENTS

DEDICATION.....	iii
ACKNOWLEDGMENT.....	iv
ABSTRACT.....	v
List of Illustrations.....	ix
List of Tables.....	xii
1 Introduction.....	1
1.1 Similar Works.....	5
1.2 Objective.....	6
1.3 Research Contribution.....	8
1.4 Outline for Thesis.....	9
2 Background.....	10
2.1 Speech Recognition.....	10
2.2 Cosine Similarity.....	14
2.3 Web Accessibility Initiative – Accessible Rich Internet Applications (WAI ARIA).....	17
2.4 Selenium Automation Testing.....	19
2.4.1 Introduction to Automation Testing.....	19
2.4.2 Selenium Testing Framework:.....	21
2.4.3 Selenium WebDriver:.....	21
2.4.4 Selenium webdriver.....	22
2.4.5 Selenium Webdriver installation.....	24
2.4.6 Browser Driver installation.....	25
2.4.7 Steps to write a selenium test case.....	27
2.4.8 Selenium test cases as a Test Case.....	29
2.4.9 WebElement in Selenium.....	32
2.4.10 Locate web elements on the web page from selenium:.....	33
2.5 Chrome Extension.....	35
2.5.1 Introduction.....	35
2.5.2 Action Types:.....	37

2.5.3	Extension scripts	39
2.5.4	Storage in Chrome Extension	41
2.5.5	Message Passing in Chrome extension	42
2.5.6	Security in Chrome extension	47
2.6	Web Accessibility definition and standards	48
2.6.1	Web Accessibility	48
2.6.2	Web Accessibility Standards	49
2.6.3	Level of conformance	50
3	VCAT Overview	52
3.1	Introduction	52
3.2	Installation steps	54
3.3	Application Design	56
3.4	The chrome browser extension module:	57
3.5	Components of the VCAT Extension	58
3.6	VCAT Tool Voice Commands:	62
3.6.1	VCAT Trigger word	65
3.6.2	VCAT voice commands	65
3.6.3	Refencing elements by Label Name:	79
3.7	VCAT Service module	80
3.7.1	Introduction	80
3.7.2	Service module design	81
3.7.3	Steps to create a selenium test case source code	82
4	Study Participation	85
4.1	Study participation process.	85
4.2	Test cases for the Study participations	85
4.2.1	Setting-up the Auto Complete Configuration Section.	85
4.2.2	Testcase 1: Registration and login	86
4.2.3	Test case 2: Google search	88
4.2.4	Test case 3: Expedia Flight Search	90
4.2.5	Test case 4: Amazon shopping cart	92

4.2.6	Test case 5: UTA Website	93
5	Experimental Results and Analysis	95
5.1	Preparticipation Survey Metrics.....	95
5.2	Result Analysis.....	103
5.3	Study participants results	103
5.3.1	Results.....	106
5.3.2	Specifications of the Server used in the Study participation:	106
6	Summary and Conclusions	107
6.1	Conclusion.....	107
6.2	Future enhancements.....	107
	Appendix A Operations that can be performed from selenium on the web elements	109
	Appendix B List of libraries used:	111
	Appendix C Links to VCAT.....	112
	References.....	113

List of Illustrations

Figure 1 People with disability trying to use a computer [1].....	1
Figure 2 Using a mouth wand to access a webpage [2].....	2
Figure 3 Using a head wand to access computer [3]	3
Figure 4 Using an eye tracker to use a computer [4]	3
Figure 5 Selenium Architecture [18]	23
Figure 6 Selenium Client version currently available [19].....	24
Figure 7 Selenium Java Driver Archive file contents.....	25
Figure 8 Selenium API JAR files.....	25
Figure 9 Current version of Chrome driver for Selenium [20].....	26
Figure 10 Execution steps of a Selenium Test case	27
Figure 11 Selenium Test Case Structure.....	31
Figure 12 VCAT Extension on Chrome web store.....	54
Figure 13 Chrome extension icon on the browser	55
Figure 14 High level diagram of the VCAT Tool.....	56
Figure 15 Flow diagram of VCAT.....	57
Figure 16 VCAT Configuration : Turn On/OFF.....	58
Figure 17 VCAT Configuration: Autocomplete	59
Figure 18 VCAT Configuration: Remote Server Endpoint	60
Figure 19 Flowchart of VCAT recognizing voice command	64
Figure 20 VCAT Select All Links command	68

Figure 21 VCAT Select All Buttons Command	68
Figure 22 VCAT Select All Text Boxes Command	69
Figure 23 VCAT Select Number Command.....	70
Figure 24 Flowchart for Select Command.....	71
Figure 25 Flowchart for Deselect Command.....	72
Figure 26 Flowchart for Click Command.....	73
Figure 27 Flowchart for Set/Enter Value Command	75
Figure 28 Message dialog displaying the generated selenium test case on the browser.....	77
Figure 29 Flowchart for referencing element by label.....	80
Figure 30 Components of the VCAT Service module.....	82
Figure 31 Flowchart for VCAT Service	84
Figure 32 Testcase 1: Registration Page.....	87
Figure 33 Testcase 1: Login page	88
Figure 34 Testcase 2: Google Search.....	88
Figure 35 Testcase 2: Google search result	89
Figure 36 Testcase 3: Expedia flight search	90
Figure 37 Testcase 3: Results on Expedia from flight search.....	91
Figure 38 Testcase 4: Amazon Shopping cart	92
Figure 39 Testcase 5: UTA website.....	93
Figure 40 Study participant consent	95

Figure 41 Study participant gender distribution	96
Figure 42 Study participant age distribution.....	96
Figure 43 Study participants highest degree received distribution.....	97
Figure 44 Study participants current UTA enrollment distribution.....	97
Figure 45 Study participants chrome browser usage experience distribution	98
Figure 46 Study participants coding experience distribution	98
Figure 47 Study participants number of lines of written code distribution	99
Figure 48 Number of Study participants who have written test cases	99
Figure 49 Study participants selenium experience distribution.....	100
Figure 50 Number of test cases written by study participants	101
Figure 51 Average Time for test case generation in Seconds.....	103

List of Tables

Table 1 Term frequency values.....	16
Table 2 Selenium web driver compatibility.....	22
Table 3 VCAT Basic browser operational commands	66
Table 4 VCAT Scroll commands.....	67
Table 5 Mapping of HTML elements in VCAT	67
Table 6 VCAT Configuration screen specific commands	78
Table 7 Test Case 1: Registration and Login.....	103
Table 8 Test Case 2: Google Search.....	104
Table 9 Test Case 3: Expedia Flight Search.....	104
Table 10 Test Case 4: Amazon Shopping Cart.....	105
Table 11 Test Case 5: UTA website navigation	105

1 Introduction

A large share of current applications assume that their users will primarily interact with the application via traditional input devices such as keyboard, mouse, or touch screen. For example, this is a common assumption made by many consumer-facing applications such as web-based applications and by many applications that are designed for software engineers such as test case generation and management tools. These applications often neglect accessibility via alternative input devices. This becomes a problem if a user cannot use [1] (Figure-1) or does not want to use one of the standard input devices.



Figure 1 People with disability trying to use a computer [1]

Currently in the market there are hardware devices for people with disabilities to help them access a computer. Some of the devices used are mouth wand [2] (Figure-2), head wand [3] (Figure-3) and eye motion sensor [4] (Figure-4)



Figure 2 Using a mouth wand to access a webpage [2]



Figure 3 Using a head wand to access computer [3]



Figure 4 Using an eye tracker to use a computer [4]

The current applications' problem of lacking accessibility is significant in practice, as many people (e.g., due to an injury or disability) cannot use or have a hard time using traditional computing input devices. This lack of accessibility can thus exclude many people from

widely used applications that are otherwise part of daily life, such as popular web pages or software engineering tools.

As a concrete example, in this paper we focus on adding accessibility to use chrome browser with voice commands and automatically generate test cases with the popular Google Chrome browser and the popular Selenium framework [6]. In a typical workflow, a software engineer

1. First uses Chrome to explore the web site application under test.
2. In a second step, the engineer writes a Java test case in the Selenium framework, to codify the actions taken in step (1).

In many cases the software engineer uses for input in both steps keyboard and mouse. While the general domain of accessibility has seen a lot of important contributions over the last decades, it is still hard today to use popular software products such as Chrome and Selenium without mouse or keyboard. Specifically, several approaches exist to automatically transcribe screen contents to voice and provide such program output to the user. However, going the opposite direction, translating from user commands to program input, seems to have less support. The likely reason for this mismatch is that going from screen to text mainly requires transcribing known program entities such as strings but going the other direction may require very complex tasks such as natural language processing, to understand user commands. Machine learning, natural language processing (NLP), and voice recognition are active areas of research. Only recent years have seen the emergence of relatively robust voice recognition systems such as Alexa [5] , Cortana [6], and Siri [7].

Since such state-of-the art systems do not yet support web browsing, now seems like a good time to revisit the software accessibility challenge.

1.1 Similar Works

There have been attempts in the recent years for adding the feature of voice commands to chrome browser. Some of the popular extensions available and the limitations when compared to VCAT are listed below:

1. LipSurf [30] - Voice Control for the Web
 - Has limited features mainly aiming at filling text.
 - No means of identifying elements by the labels.
 - Does not generate test cases.
2. Click by Voice [31]
 - Has feature to only open links.
 - Does not generate test cases
3. Hands Free for Chrome [32]
 - Navigating webpage has a lot of steps and makes it difficult to remember the entire procedure. Commands sequence is very technical and difficult for layman.
 - No means of identifying elements by the labels.
 - Does not generate test cases.
4. Voice Actions for Chrome (beta) [33]
 - Used to only operate the browser and not helpful in webpage navigation.
 - Does not generate test cases.

5. Dragon Naturally Speaking from Nuance [37]:

- Primarily aimed at native windows applications.
- Has limited features for web browsing.
- Paid software.

VCAT has the following advantages when compared to the above extensions:

1. Availability of the option to refer the element name by label.
2. Availability of key value auto completion configuration.
3. Availability of the feature to capture all the steps executed by the user and generate the java selenium test case.

1.2 Objective

To address this challenge, this paper describes the Voice Controlled Accessibility and Testing tool (VCAT), which accepts user voice commands to navigate complex web sites on a Google Chrome browser. The system works as a browser extension, logs all user actions, and transcribes them as Selenium test cases. In an initial evaluation, we found that by generating test cases from logged user voice commands, VCAT can significantly reduce the time a traditional expert user needs to create Selenium test cases. This bodes well for the target scenario in which a user that requires accessible software browses the web with Chrome and issues voice commands, enabling such a user to perform parts of routine software engineering work.

To summarize, this work makes the following contributions.

- The VCAT scheme generates test cases of web-based applications entirely via voice commands.
- In an initial evaluation, VCAT reduced the time required to create Selenium test cases.

The VCAT tool was presented at the ICSE 2018 [8] conference held in Sweden, in the poster presentation section. The poster presentation was aimed at presenting the technological feasibility of using speech commands to perform web page navigation and selenium code generation. Results from the proof of concept were used in the poster presentation [9]. At the conference, several attendees provided favorable interest in the VCAT tool, and feedback was also provided by some of the attendees. One key element of the VCAT tool, specifically numbering the elements on the web page, was implementing as a result of the poster presentation.

(N. P. Kasaghatta Ramachandra and C. Csallner, "Poster: Testing Web-Based Applications with the Voice Controlled Accessibility and Testing Tool (VCAT)," *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, Gothenburg, 2018, pp. 208-209.)

The VCAT tool was also presented at the University of Texas at Arlington, College of Engineering, on Annual Innovation Day, 2018 [10].

1.3 Research Contribution

VCAT tool has the following research contributions:

1. A Chrome browser extension that leverages the Natural language processing capabilities to add accessibility to the browser. This enables people with disabilities to use web browsers with voice commands.
2. Automatic Java selenium code generation module is a novel method in automation testing code generation that helps in reducing overall coding time and help people with disabilities take up automation testing activities easily.

1.4 Outline for Thesis

This thesis is organized into the following chapters respectively:

Chapter 1 – Introduction: This chapter explains the problems in accessibility of websites and the applicability of the VCAT tool to minimize this.

Chapter 2 – Background: This chapter presents the background and concepts of accessibility, automation testing using selenium, Natural Language Processing (NLP), chrome browser extension.

Chapter 3 – VCAT Overview: This chapter presents design and implementation of the VCAT tool, test set-up and execution procedure in this study.

Chapter 3 – Study Participation: This chapter presents test set-up and execution procedure in this study.

Chapter 5 – Experimental Results and Analysis: This chapter presents the results from the using of VCAT tool by the test subjects and the analysis of the results.

Chapter 6 – Summary and Conclusions: The findings of this research are summarized, and the conclusions are presented.

2 Background

2.1 Speech Recognition

Speech Recognition is a form on Natural Language Processing which involves recognition of speech and translating the speech into text. Speech recognition is an interdisciplinary since it involves linguistics, mathematics and computer science algorithms. There are different approaches for speech recognition [8]:

1. Simple pattern matching
2. Pattern and feature analysis
3. Language modeling and statistical analysis
4. Artificial Intelligence based analysis

1. Simple pattern matching: This is the simplest type of speech recognition that is primarily used in automated call centers over the IVR. The system does not understand the meaning, syntax or grammar of the input speech. A standard set of voice inputs is matched to certain actions like keys on the number pad. A typical example would be when calling a banking call center to speak the account number instead of keying in number on the phone keypad. The system is programmed to match the voice samples of the numeric digits to the respective numbers. Since the ten different numbers in the decimal system are distinct in pronunciation, the accuracy of identifying the correct number is considerably high. But people speak in different tone speed and accent and

in case the system fails to recognize any matching pattern of the input speech, then the call is usually transferred to a human operator.

2. Pattern and feature analysis: In this approach the linguistic nature of speech and mathematics is applied for speech recognition. When we look at any language it has some fundamental common features associated with it. The fundamental unique notation in any language is called as an *alphabet*. Alphabets are joined together in a specific manner to form *words*. Words along with empty spaces between them form *sentences* which would convey the actual thought of the speaker into text. When we speak our voice generates chunks of sounds that are called as *phones* and the words are comprised of basic blocks of chunks of sounds called *phonemes*. In any language there are definite set of phonemes defined from which any word is formed in that language. English language has 46 distinct phonemes. Hence any word that is spoken in English language must comprise of one or more of these 46 phonemes.

In the pattern and feature analysis approach the analog speech signal is first converted into a digital signal using an Analog-Digital converter. Then using the mathematical **Fast Fourier Transform (FFT)** the digital signal is converted into a *spectrogram*. The signal is then broken down into acoustic frames each of them lasting from $1/25$ to $1/50$ of a second. Then the chunks are further processed to extract the phonemes in each of the word. Words are identified in a sentence by recognizing silence in the gap between the words in a sentence. Once every word is broken down into the composing phonemes, a dictionary called as *phonetic dictionary* is used to identify the actual word. The phonetic dictionary comprised of a mapping of words and the phonemes it is

comprised of. The dictionary is required to be updated with all the words and its phonemes sequence. In case the speech has a specific word that is not mapped in the phonetic dictionary it will not be recognized.

3. Language modelling and statistical analysis: Not all people speak in the same way and use the same words in a sentence. Even through the language is same, due to difference in the ethnic origin and/or country of origin, there can be variation in the manner the language is spoken which is called as accent. For example, the accent of an English Native American would be different from the accent of a person from Indian origin. Accent can interfere in the way the words are pronounced and hence the outcome of speak recognition system recognizing the word efficiently could vary. There could also be instances when the same person is speaking the same word in different ways at different instances. The English language has many words that sound very similar. Example the words to, too and two sound the same but have different meanings. Hence pattern and feature analysis of the speech input could fail many times.

When we look at a language, it is not just a mix and match of words and letters. There are certain rules known as a grammar that must be used when we speak any language. For example, in English language we must use the word “an” before any word that is starting with a vowel and “a” before a word starting with a consonant. Thus, if we see the word an then it would be highly probable that the next word would start with a vowel unless the speaker makes a mistake. The concept of grammar combined with the mathematical concepts of Hidden Markov model and Viterbi Algorithm to get a much more efficient speech recognition system. The Hidden Markov model uses an approach

to predict the most likely outcome of the next state based upon the current state. This approach effectively improves the efficiency of the speech recognition system.

4. Artificial intelligence-based analysis: Artificial intelligence leverage the power of computing and learning capabilities of the systems using feedback mechanisms. One of the main applications of Artificial Intelligence has been in the field of speech recognition. With better access to internet and cloud computing combined with more efficient AI models, speech recognition has become more efficient and readily available across multiple platforms and devices. Products like Alexa from Amazon, Google Assistant from Google and Siri from Apple and a wide range of third-party implementations, speech recognition has penetrated newer domains across many industries.

2.2 Cosine Similarity

Cosine Similarity Algorithm is used to find the amount of similarity between two given strings or words [9]. The input to the algorithm is two strings or words to be compared for similarity and the output is a score between 0 and 1. The scores is calculated by quantifying the similarity of sequences by treating the strings or words as vectors and calculating their cosine. A score of 0 indicates that there is no similarity between the two strings or words and a score of 1 indicates that the given two strings or words are same.

Cosine Similarity Formula:

The formula used to calculate cosine similarity is given by the below equation:

$$similarity(\vec{a}, \vec{b}) = \cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

The above equation can be re-written as,

$$\frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n (a_i)^2} \times \sqrt{\sum_{i=1}^n (b_i)^2}}$$

Steps to calculate Cosine Similarity:

1. Calculate the Term Frequency of every letter in the given two strings or words in form of two vectors. Term frequency is the number of times every unique word appears in the two strings. In case of words, the term frequency of every unique character is tabulated
2. Get the dot product of the two vectors.
3. Calculate the magnitude of the two vectors.
4. Divide the dot product of the vectors with the product of the magnitude of the two vectors.

Consider the two words: “Click” and “Blink”

- a. Click
- b. Blink

Step1: The term frequencies are calculated as follows:

	Letter	Click	Blink
1	B	0	1
2	C	2	0
3	L	1	1
4	I	1	1

5	N	0	1
6	K	1	1

Table 1 Term frequency values

Vector a : (0,2,1,1,0,1) and Vector b: (1,0,1,1,1,1)

Step 2: Dot products of the two vectos:

$$(\text{Vector a}).(\text{Vector b}) = 0 \times 1 + 2 \times 0 + 1 \times 1 + 1 \times 1 + 0 \times 1 + 1 \times 1 = 3$$

Step 3: Calculate the magnitude of each vector:

$$\|\text{Vector a}\| = \text{Sqrt}(2^2 + 1^2 + 1^2 + 1^2) = 2.645$$

$$\|\text{Vector b}\| = \text{Sqrt}(1^2 + 1^2 + 1^2 + 1^2 + 1^2) = 2.236$$

Step 4: Calculate the Cosine Value using the formula:

$$3 / (2.645) \times (2.236)$$

$$= 0.507$$

Hence the given two words are 50% similar.

2.3 Web Accessibility Initiative – Accessible Rich Internet Applications (WAI ARIA)

According to W3C technical specification documentation, “WAI-ARIA, the Accessible Rich Internet Applications Suite, defines a way to make Web content and Web applications more accessible to people with disabilities. It especially helps with dynamic content and advanced user interface controls developed with Ajax, HTML, JavaScript, and related technologies.” [10].

People with certain disabilities use Assistive Technologies to help them use applications. The assistive technologies can present the underlying application in a way that the person can use it without having the disability to stop them. One example is using a screen reader for users with blindness. Screen readers readout the description of the element on which the user would either move the mouse, click the mouse or use the keyboard. A disabled user who cannot use a mouse or touch screen would prefer to use the keyboard only to fill up online forms.

The assistive technologies look out for special attributes that are used in every element to determine the kind of element it is and what is the purpose of the element on the web page. For example, say there is a text box with a label that describes the text box next to it. As per the WAI-ARIA standards, **aria-labeledby** attribute must be used with the label to associate the label with its target text box. Though visually, looking at the placing of the text box and the label we may associate its meaning, Assistive technologies will have no means to come to this judgment without the use of WAI-ARIA tags. If a blind user clicks on a text field the screen reader would look out for the label associated with the text box

and use it to describe the text box. If there is no use of aria tags, then the screen reader would just describe as a text box and the blind user would have no means to understand what the purpose of the text box on the web page is.

WAI-ARIA tags are extremely important in situation where the developer uses nonstandard elements on the webpage. For example, a developer may choose to use an image to look and function like a button. A screen reader without the WAI-ARIA tags would only identify it as an image instead of a button. But with the use of proper WAI-ARIA tags, the screen reader would be able to describe the image as a button as intended and thereby giving more clarity to the user about the element and its role in the webpage.

There are two main categories of WAI-ARIA tags as per the W3C specifications:

1. **WAI-ARIA Roles:** The role attribute is used to describe the intended role of the attribute. For example, if a list item “li” element is used to create a menu item, the aria role "menuitem” would indicate that it is a menu item. More information about the different roles can be found here [11].
2. **WAI-ARIA States and Properties:** These tags are used to describe the current state and property of the element. For example, aria-checked attribute would be used to indicate if a checkbox is checked or unchecked. More information about the various states and properties can be found here [12].

2.4 Selenium Automation Testing

2.4.1 Introduction to Automation Testing

Application testing refers to the testing of the overall software application behavior or both front end and back end functionalities. Testing encompasses end to end functional testing of the application. This helps the team to ensure the software quality of the application and at the same time, it reduces rework, security breaches and application failure issues.

The two important steps any organization must take care is verification and validation.

According to the IEEE SWEBOK V3 section 10.2.2 [13]

1. Verification: "... ensure[s] that the product is built correctly." Is the product built to its requirements?
2. Validation: "... ensure[s] that the right product is built—that is, the product fulfills its specific intended purpose." Are the requirements correct?

There are two ways of testing is performed in every organization. It could be one or a combination of the both:

1. Manual testing: A testing team manually document a detailed test plan in collaboration with the Business Analyst and the development team. Once a test plan is in place the manual testing team go over every scenario called as "*Test cases*". An excel sheet is a maintained to document the expected and the actual result of the test case execution. If the actual result matches the expected result, the test case is considered pass else a fail. The results are documented for every test case and the overall Pass v/s Fail denote the overall health of the application.

2. Automation testing: Automation testers use programming frameworks to automate the process of testing to achieve faster, consistent, efficient and reusable test coverage. Every test case is written in form of a test program. When the test program is executed successfully, the test case is considered a pass. If the program fails or does not meet the intended end result (by using the assert statement and it returns false during execution), the test case is considered as false. The process of automation testing is faster and more reliable. Stages such as Integration testing and smoke testing can be completed quickly since many test cases can be run repeatedly achieving higher efficiency in test coverage. Some of the top automation testing frameworks used in the current industry are Selenium [14], Soap UI [15] and QTP [16].

2.4.2 *Selenium Testing Framework:*

Selenium is an opensource testing framework used to test web applications. Selenium is browser and platform independent and hence the most popular automation testing framework [17].

Primarily, Selenium was originally created by Jason Huggins in 2004. He created a JavaScript program that would automatically control the browser's actions called "*JavaScriptTestRunner*", Observing the potential in JavaScriptRunner to help automate any web applications, he made it open-source which was later re-named as Selenium Core.

2.4.3 *Selenium WebDriver:*

Simon Stewart created WebDriver in 2006 was the first cross-platform testing framework that could control the browser from the OS level. It implements a more modern and stable approach in automating the browser's actions. It controls the browser by directly communicating with it.

The supported languages are the same as those in Selenium RC.

- Java
- C#
- PHP
- Python
- Perl
- Ruby

2.4.4 Selenium webdriver

Selenium Webdriver is an automation framework designed to accept command and then send to the browsers to be executed. The important aspect of the selenium webdriver is that has browser specific drivers. Selenium webdrivers are compatible with the following browser, operating system and programming languages:

Browsers	Mozilla Firefox, Chrome, Internet Explorer, Safari and Opera,
Operating System	Windows, Mac OS, Linux and Solaris
Programming languages	Java, C#, python, PHP, Perl and Ruby

Table 2 Selenium web driver compatibility

The architecture of the selenium webdriver [18] is divided into four layers:

1. Selenium Language Bindings
2. JSON Wire Protocol
3. Browser Drivers
4. Real Browsers

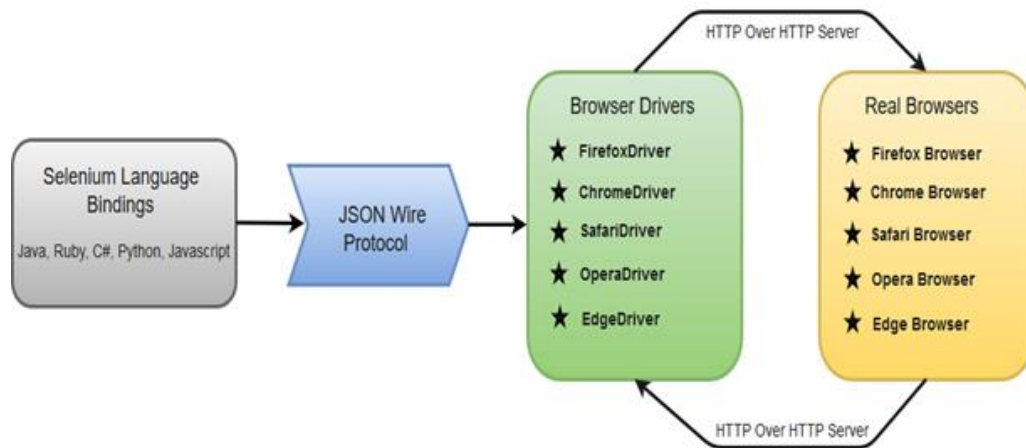


Figure 5 Selenium Architecture [18]

1. Selenium Language Bindings: Selenium code can be written in different programming languages. There are separate language binding drivers for each of the languages. For example, if java is the programming language then the Java-Selenium Language binding driver must be used.
2. JSON wire protocol: JavaScript Object Notation -JSON is an industry standard for exchanging data over the web. JSON is a connecting bridge between the selenium code and the driver. This is a form of a client-server interaction within the selenium framework. The advantages of using JSON are:
 1. Test cases can be written in testers choice of programming language.
 2. Selenium test cases can be run locally or remotely on a cloud.
 3. Different browser drivers can be created keeping the same standards.
3. Browser Drivers: The browser driver is used as an encapsulation for connecting to an actual browser and executing the commands. Once the browser driver receives every command in the form of a HTTP request. The browser driver then sends the

command to execute the step to an actual browser is sent via an HTTP server. The response from the actual browser is then propagated back to the browser driver via the HTTP server and then to the automation script.

4. Real Browsers: The browsers on which the script is intended to be executed.

2.4.5 Selenium Webdriver installation

The steps to install the webdriver is as follows:

1. The latest versions of webdrivers are available on the site [19].
2. Pick the driver to download as per the programming language:

Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

While language bindings for [other languages exist](#), these are the core ones that are supported by the main project hosted on GitHub.

Language	Client Version	Release Date			
Java	3.141.59	2018-11-14	Download	Change log	Javadoc
C#	3.14.0	2018-08-02	Download	Change log	API docs
Ruby	3.14.0	2018-08-03	Download	Change log	API docs
Python	3.14.0	2018-08-02	Download	Change log	API docs
Javascript (Node)	4.0.0-alpha.1	2018-01-13	Download	Change log	API docs

C# NuGet

Figure 6 Selenium Client version currently available [19]

3. In the case of Java programming language, the downloaded file is a zipped file contains the jar files **client-combined-3.141.59.jar** and **client-combined-3.141.59-sources.jar** required to configure the selenium webdriver

Name	Size	Packed	Type	Modified	CRC32
Local Disk					
libs			File folder	2/1/1985 12:00 ...	
client-combined-3.141.59.jar	1,527,879	1,438,576	Executable Jar File	2/1/1985 12:00 ...	2A698E43
client-combined-3.141.59-sources.jar	529,743	477,209	Executable Jar File	2/1/1985 12:00 ...	039F3E43
CHANGELOG	121,465	43,027	File	2/1/1985 12:00 ...	E9AC3E9A
LICENSE	11,365	3,961	File	2/1/1985 12:00 ...	2D9D0496
NOTICE	89	78	File	2/1/1985 12:00 ...	EE63D047

Figure 7 Selenium Java Driver Archive file contents

Name	Size	Packed	Type	Modified	CRC32
Local Disk					
byte-buddy-1.8.15.jar	2,987,269	2,613,396	Executable Jar File	2/1/1985 12:00 ...	87278709
commons-exec-1.3.jar	54,423	47,600	Executable Jar File	2/1/1985 12:00 ...	A8AC0A14
guava-25.0-jre.jar	2,738,171	2,438,570	Executable Jar File	2/1/1985 12:00 ...	C448AB79
okhttp-3.11.0.jar	413,639	391,155	Executable Jar File	2/1/1985 12:00 ...	D153BDE1
okio-1.14.0.jar	85,756	82,065	Executable Jar File	2/1/1985 12:00 ...	0D3A89BA

Figure 8 Selenium API JAR files

4. Add the jar files into the class path of the java project.

2.4.6 Browser Driver installation.

Every selenium automation code must indicate the location to the browser driver location.

The following steps indicate how to configure the chrome browser driver in a java selenium automation code:

1. Download the chrome driver from the following location [20].

2. Download the correct version of the browser driver according to version of the chrome browser installed on the computer:

Current Releases

- If you are using Chrome version 76, please download [ChromeDriver 76.0.3809.25](#)
- If you are using Chrome version 75, please download [ChromeDriver 75.0.3770.90](#)
- If you are using Chrome version 74, please download [ChromeDriver 74.0.3729.6](#)
- If you are using Chrome version 73, please download [ChromeDriver 73.0.3683.68](#)
- For older version of Chrome, please see below for the version of ChromeDriver that supports it.

Figure 9 Current version of Chrome driver for Selenium [20]

3. The downloaded file is a zip file. Extract the contents to find the file *chromedriver.exe*.
4. For the selenium test cases to execute on a chrome browser, the location to the chromedriver.exe must be set to *webdriver.chrome.driver* flag. This is achieved by setting the value at the start of the selenium test case as follows:

```
System.setProperty("webdriver.chrome.driver", "D:\\ChromeDriver\\chromedriver.exe");
```

2.4.7 Steps to write a selenium test case

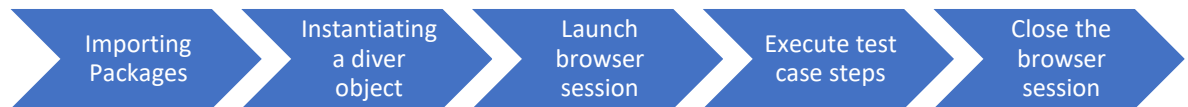


Figure 10 Execution steps of a Selenium Test case

1. Importing Packages:

The following two base packages must be imported for every selenium test case:

- a. **org.openqa.selenium.***- contains the WebDriver class needed to instantiate a new browser.
- b. **org.openqa.selenium.chrome.ChromeDriver** - A WebDriver implementation that controls a Chrome browser running on the local machine.

2. Instantiating a driver object: A driver object is instantiated by executing the following line. Once the driver object is instantiated every step of the selenium test case is passed to the driver object:

```
WebDriver webDriver = new ChromeDriver();
```

3. Launch browser session: The browser session is launched by invoking the `get()` method which accepts a string argument which is the URL of the website :

driver.get("URL");

4. Execute the test case steps: The actual steps of the test case are now executed once the browser session is launched. Details of the selenium api write test cases is discussed in the following section:
5. Close the browser session: Once all the test case steps has been executed the browser session has to be closed by invoking the close() method:

driver.close();

2.4.8 *Selenium test cases as a Test Case*

Since every selenium class is intended to be a test case, we can leverage some of the lifecycle methods of a Testcase in a selenium test case by extending the Test Case class.

The following lifecycle methods can be used:

1. **@BeforeClass:** This annotation is used for a method that must be executed once for every test case class before any other method is executed. In this method we can configure the path to the selenium chromedriver location
2. **@AfterClass:** This annotation is used for a method that must be executed once for every test case class and the last method. The method can be used to perform any clean up.
3. **@Before:** This annotation is used for a method that must be executed before every test case execution. The instantiation of the driver object can be performed here.
4. **@After:** This annotation is used for a method that must be executed after every test case. The closing of the browser session can be performed here.
5. **@Test:** This annotation indicates that the method is a test case.

A sample selenium test case can be thus structured as follows:

```
@RunWith(BlockJUnit4ClassRunner.class)

public class ChromeTest extends TestCase {

    private static ChromeDriverService service;

    private WebDriver driver;

    @BeforeClass

    public static void createAndStartService() {

        service = new ChromeDriverService.Builder()

            .usingDriverExecutable(new File("path/to/my/chromedriver"))

            .usingAnyFreePort()

            .build();

        service.start();

    }

    @AfterClass

    public static void createAndStopService() {

        service.stop();

    }

}
```

```
@Before

public void createDriver() {

    driver = new ChromeDriver();

}

@After

public void quitDriver() {

    driver.quit();

}

@Test

public void testGoogleSearch() {

    driver.get("URL");

    // rest of the test...

}

}
```

Figure 11 Selenium Test Case Structure

2.4.9 Webelement in Selenium

Forms essential html mechanism for websites to connect the users and the business process. Web forms have different GUI elements like Text boxes, Password fields, Checkboxes, Radio buttons, dropdowns, file inputs, etc. [21].

In order to test the proper functioning of the application we need to test the functionality of the web form as per the requirements of the application. Selenium in accordance to the html standards offers web elements to identify the various html web elements that are available. The web elements also offer the respective actions or events to be triggered on the web form to test the required functionalities.

WebElement is a superclass for every type of an element. The actions that can be triggered on a webelement would be dependent on the underlying element. The different types of webelements are:

1. Textbox: It is a basic text control that enables a user to type a single line of text. It could be of type plain text, password, phone number or email.
2. Link: *Link* commonly known as a *hyperlink* and connects one web page to another. It allows the user to click on the link to open the underlying target webpage.
3. Button: This represents a clickable button, which can be used in forms and places in the document that needs a simple, standard button functionality. Submit, button and reset are the standard type of button available to be used in a form.

4. Image: Images as the name goes is used to display pictures on a webpage. Images could also be used as a link or a button by adding additional functionalities on an image.
5. Text area: It is used to enter multiple lines of text.
6. Checkbox: This is a selection box that can be toggled identifies by checked or unchecked by the user to indicate a selection or non-selection choice respectively.
7. Radio button: It is a group of elements which is allows the user to choose only one predefined set of mutually exclusive options.
8. Dropdown list/Select: It is a drop-down list of values from which a user can select one or more values from.

2.4.10 *Locate web elements on the web page from selenium:*

Selenium provides functionalities that help in locating the element on a web page. It makes use of element locators [22]. In order to identify WebElements accurately and precisely, Selenium makes use of different types of locators, namely:

1. Id locator: The most popular way to identify web element is to use ID. ID's are considered as the safest and fastest locator option and should always be the first priority among multiple locators.
2. Name locator: This is also an effective way to locate an element which has a name attribute. With this strategy, the first element with the value of the name attribute

will be returned. If no element has a matching name attribute, then a **NoSuchElementException** will be raised.

3. LinkText & Partial linkText : You can identify the hyperlinks on a web page using this linkText. It can be determined with the help of an anchor tag (<a>). To create the hyperlinks on a web page, you can use the anchor tags followed by the link text. In some cases, you may need to find links by a portion of the text in a linkText element. In such situations, you can use Partial Link Text to locate elements.
4. CSS Selector: CSS is mainly used to provide style rules for the web pages, and you can use it for identifying one or more elements in the web page. The CSS selector is always the best possible way to locate complex elements in page.
5. XPath: XPath is a language to query XML documents. XPath is an important strategy to locate elements in selenium. It also consists of a path expression along with some conditions. Here, you can easily write an XPath script/query to locate any element in the webpage.

2.5 Chrome Extension

2.5.1 Introduction

Chrome extensions are user build software components that are installed on a chrome browser to add additional functionalities to the browser [23]. A chrome browser extension is build using the standard Chrome extension api in conjunction with HTML, JavaScript and CSS.

The structure of a chrome extension is as follows:

1) Manifest File: Every extension has a JSON [24] formatted manifest file, named manifest.json, where the configurations of the chrome extension are defined. The entries in the manifest file are divided into required, recommended and optional entries.

a) Required Entries: These are the entries that must be present in the manifest file of every chrome extension:

i) manifest_version: Number specifying the version of the manifest file format your package requires. As of Chrome 18, developers *should* specify **2**.

```
"manifest_version": 2
```

ii) name: The name field is the identifier of the extension. It can have a maximum length of 45 characters. It is displayed in the install dialog, Extension management console and the Chrome web store.

```
"name": "My Extension"
```

iii) version: One to four dot-separated numbers are used to identify the version of the extension. The rules applicable for versioning are they must be between 0 and 65535, inclusive, and non-zero integers can't start with 0. For example, 99999 and 032 are both invalid.

```
"version": "1"
```

b) Recommended Entries: These entries are good to have but not mandatory in every extension:

i) default_locale : Used to specify the language short form for localization.

```
"default_locale": "en"
```

ii) Description: A plain text string no more than 132 characters that describes the extension.

```
"description": "A plain text description"
```

iii) Extension icon: The extension icon is the image that is used to identify the extension. It is configured in the manifest file. Every extension must have the following configuration in the manifest file:

```
"icons": { "16": "icon16.png",  
           "48": "icon48.png",  
           "128": "icon128.png" }
```

There should always be a 128x128 icon which is used during the installation of the extension and by the Chrome Web Store. Extensions should also provide a 48x48 icon, which is used in the extension's management page

(chrome://extensions). There could also be a 16x16 icon to be used as the favicon for an extension's pages.

- c) Action type entries: Used to indicate the type of the action the extension is used for could be either of one of the following or none:

```
"browser_action": {...},
```

```
"page_action": {...},
```

- d) Optional entries: These entries are good to have but optional and not mandatory. It is used to provide extra functionalities to the extension depending on the requirements of the specific extension.

2.5.2 Action Types:

There are two possible types of extension on a high level identified as a “*Browser Action*” extension or a “*Page Action*” extension.

Browser Action: The extensions of this type can be used on all the pages that are opened in the browser and not limited to any specific websites. The browser action icon is displayed on the chrome browser toolbar next to the address bar. The extension can be configured as a browser action by the following entries in the manifest file:

```
"browser_action": {
```

```
  "default_icon": { // optional
```

```
    "16": "images/icon16.png", // optional
```

```
    "24": "images/icon24.png", // optional
```

```
    "32": "images/icon32.png" // optional
```

```

    },
    "default_title": "title", // optional; shown in tooltip
    "default_popup": "popup.html" // optional
  }

```

Page Action: the page action extension is used to add specific functionalities to certain websites. Unlike a browser action extension, the page action extension icon is displayed in the address bar in the browser and not in the toolbar. The page action icon gets visible only when the intended website is launched in the browser. The extension can be configured as a page action by the following entries in the manifest file:

```

"page_action": {
  "default_icon": { // optional
    "16": "images/icon16.png", // optional
    "24": "images/icon24.png", // optional
    "32": "images/icon32.png" // optional
  },
  "default_title": "title", // optional; shown in tooltip
  "default_popup": "popup.html" // optional
}

```

2.5.3 *Extension scripts*

There are two types of scripts used in a chrome extension to handle events. Background script and Content script. An extension can use either of them, both or none as per the functional requirement.

2.5.3.1 Background Script

The background script is used to handle specific events and respond to them. The required events must be registered in the background script listener. The background script is loaded when required and then unloaded when it is not in use. A background script can get active and fired for example when the event that the script is listening to is dispatched or when a message is sent to the background page. The background script remains dormant until the event that it is listening to is fired, perform the action and then is unloaded.

The first step in using a background script is to register it in the manifest file as follows:

```
"background": {  
  "scripts": ["background.js"],  
  "persistent": false  
}
```

The script attribute indicates the name of the background JavaScript file. It accepts an array as a value and thus enabling the extension to register multiple background pages. The persistent attribute must always be set to false for a background script.

2.5.3.2 Content Script

The content scripts are used to execute code on the actual content of the page. The content script unlike the background script has access to the DOM structure of the page. This enables the content script to be able to read the contents of the webpage, make changes to the contents of the page if required and communicate information about the webpage to the extension via messaging.

A content script can be used in a chrome extension by configuring the corresponding JavaScript file in the manifest file:

```
"content_scripts": [  
  {  
    "matches": ["http://*. com/*"], //mention the url pattern that would be  
    used to match to execute the content script through the extension  
    "css": ["myStyles.css"], // to provide the extension specific styling  
    "js": ["contentScript.js"] // the content script  
  }  
]
```

Content scripts can access the DOM structure of the webpage, retrieve data in the webpage, modify the content in the webpage, apply or remove styling in the webpage.

2.5.4 Storage in Chrome Extension

Chrome browser extensions provides options to store persistent data within the extension.

To enable data storage, the extension must be granted storage permission in the manifest file as follows:

```
"permissions": [
  "storage"
]
```

Data can be stored in the extension in two ways:

1. Local mode: In this mode the data is stored locally in the browser. The data can be stored and retrieved in the scripts as follows:

Store the data:

```
chrome.storage.local.set({key: value}, function() {
  console.log('Value is set to ' + value);
});
```

Retrieve the data:

```
chrome.storage.local.get(['key'], function(result) {
  console.log('Value currently is ' + result.key);
});
```

2. Sync mode: In this mode the data that is stored will be synced to any chrome browser that the user has logged in. In case the user has not logged into the chrome browser or is not online, then the data is stored in local mode. The data is then synced as soon as the user comes online next time. A total of 102,400 bytes of data (measured by the JSON stringification of every key and value length) can be stored in the sync mode. The data can be stored and retrieved in the scripts as follows:

Store the data:

```
chrome.storage.sync.set({key: value}, function() {  
    console.log('Value is set to ' + value);  
});
```

Retrieve the data:

```
chrome.storage.sync.get(['key'], function(result) {  
    console.log('Value currently is ' + result.key);  
});
```

2.5.5 *Message Passing in Chrome extension*

Since the content scripts are executed on the webpage and not within the extension, it may be required to communicate between the content script and the extension. This is achieved through message passing. The message passing can be two ways from the content script to the extension and vice versa. Either side can listen for message dispatched from the other

end and respond back. The content of the message request and response should be a valid JSON object.

There are two types of message passing in chrome extension:

1. One-time request: The one-time request is a short-lived connection that is established between the content script and the background script. The sequence of events in a one-time request are:
 - a. The sender initiates a communication with the receiver by sending a one-time JSON serializable message. When sending a message from the content script the `sendMessage()` method must be used as follows:

```
chrome.runtime.sendMessage({key: "value"}, function(response) {  
  // perform some action here when the response is received  
});
```

When sending a message from the extension to the content script the `sendMessage()` method must be invoked on the active tab as follows:

```
chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {  
  chrome.tabs.sendMessage(tabs[0].id, {key: "value"}, function(response)  
  {  
    // perform some action here when the response is received  
  });
```

```
});
```

- b. Once the `sendMessage()` method is invoked a communication channel is established and, on the receiver, the message can be received. Responding back to the sender with a response is optional for the receiver. The receiving of the message on the receiver is same for both content script and the extension by adding a listener to `onMessage` as follows:

```
chrome.runtime.onMessage.addListener(  
  function(request, sender, sendResponse) {  
    // perform your action here and send response if required  
    sendResponse({key: "value"});  
  });
```

2. Long lived connections: To have a conversation that lasts longer than a single request and response chrome extension api provides a framework called as the long-lived connection. In this case, we need to open a long-lived channel from the content script to the extension using **`runtime.connect`** . The channel can optionally have a name, allowing the extension to distinguish between different types of connections. When establishing a connection, each end is given a **`runtime.Port`** object which is used for sending and receiving messages through that connection.

To open a port from the content script the basic steps are as follows:


```

var port = chrome.runtime.connect({ name: "portname" });

port.postMessage({ key: "value" });

port.onMessage.addListener(function(msg) {

  if (msg == .....)

    port.postMessage({ reply: "value1" });

  else if (msg...)

    port.postMessage({ reply: "value2" });

});

```

In order to handle incoming connections, we need to set up a **runtime.onConnect** event listener which is the same from a content script or an extension page. An example to respond to incoming connections:

```

chrome.runtime.onConnect.addListener(function(port) {

  console.assert(port.name == "portname");

  port.onMessage.addListener(function(msg) {

    if (msg.key == "value")

      port.postMessage({ key: "value" });

    else if (msg.reply == "value1")

      port.postMessage({ key: "value" });

    else if (msg.reply == "value2")

```

```
port.postMessage({key:"value"});  
});  
});
```

2.5.6 *Security in Chrome extension*

Since the chrome extension provides advanced browser level api access to the background script and the content script the obvious question is about the security of the web applications.

One important design of the chrome extension is that the scripts get to execute in isolated worlds within the chrome browser. This ensures that the JavaScript objects within the webpages would not be able to access or execute the code that is present in the scripts of the extension. Thus, there would no way be possible for any script on the webpage to get access to privileged browser level api calls. Another important feature of this isolated worlds is that one content script will not bel able to access the data or the functionality of another content script. This ensures the safety of data being accessed from cross extensions. Thus, with the feature of isolated worlds, chrome extensions ensure security in the browser.

2.6 Web Accessibility definition and standards

2.6.1 *Web Accessibility*

According to the website of w3c.org [25] Web accessibility is defined as:

Web accessibility means that websites, tools, and technologies are designed and developed so that people with disabilities can use them. More specifically, people can:

1. perceive, understand, navigate, and interact with the Web
2. contribute to the Web

Web accessibility encompasses all disabilities that affect access to the Web, including:

1. auditory
2. cognitive
3. neurological
4. physical
5. speech
6. visual

Web accessibility also benefits people *without* disabilities, for example:

1. people using mobile phones, smart watches, smart TVs, and other devices with small screens, different input modes, etc.

2. older people with changing abilities due to ageing
3. people with “temporary disabilities” such as a broken arm or lost glasses
4. people with “situational limitations” such as in bright sunlight or in an environment where they cannot listen to audio
5. people using a slow Internet connection, or who have limited or expensive bandwidth

2.6.2 *Web Accessibility Standards*

Web accessibility depends on several components interacting with each other that could include the website content, design of the website, browsers, platforms and hardware. The **World Wide Web Consortium (W3C)** is an international community that develops open standards to ensure the long-term growth of the Web. The W3C Web Accessibility Initiative (WAI) publishes the current technical standards, guidelines, techniques and evaluation methodologies for achieving the web accessibility for products in the market. The WAI published accessibility standards known as the Web Content Accessibility Guidelines (WCAG) is an internationally accepted standard: **ISO/IEC 40500**.

WCAG has defined a total of 13 guidelines categories into four categories known as the *Four Principles of Accessibility* Perceivable, Operable, Understandable and Robust abbreviated in general as the **POUR** principles. As stated in the WCAG2.1 website the POUR principles are defined as follows:

1. **Perceivable** - Information and user interface components must be presentable to users in ways they can perceive.

This means that users must be able to perceive the information being presented (it can't be invisible to all their senses)

2. **Operable** - User interface components and navigation must be operable.

This means that users must be able to operate the interface (the interface cannot require interaction that a user cannot perform)

3. **Understandable** - Information and the operation of user interface must be understandable.

This means that users must be able to understand the information as well as the operation of the user interface (the content or operation cannot be beyond their understanding)

4. **Robust** - Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.

This means that users must be able to access the content as technologies advance (as technologies and user agents evolve, the content should remain accessible)

2.6.3 *Level of conformance*

For each of the guideline there is also defined a testable *success criterion* [26] that are at three levels:

1. **Level A:** For Level A conformance (the minimum level of conformance), the [Web page satisfies](#) all the Level A Success Criteria, or a [conforming alternate version](#) is provided.
2. **Level AA:** For Level AA conformance, the Web page satisfies all the Level A and Level AA Success Criteria, or a Level AA conforming alternate version is provided.
3. **Level AAA:** For Level AAA conformance, the Web page satisfies all the Level A, Level AA and Level AAA Success Criteria, or a Level AAA conforming alternate version is provided.

3 VCAT Overview

3.1 Introduction

According to the statistics on interactiveaccessibility.com [27], 57 MILLION AMERICANS HAVE A DISABILITY. As stated in their website, “According to the new figures released by the Census Bureau on July 25, 2012, 56.7 million Americans (18.7% of the U.S. population) have some type of disability and out of this number, an estimated 38.3 million (12.6%) have a severe disability.

Consider impairments that impact accessibility of online websites, applications, and documents. This survey estimates the number of people with specific impairments as follows:

1. 19.9 million (8.2%) have difficulty lifting or grasping. This could, for example impact their use of a mouse or keyboard.
2. 15.2 million (6.3%) have a cognitive, mental, or emotional impairment.
3. 8.1 million (3.3%) have a vision impairment. These people might rely on a screen magnifier or a screen reader or might have a form of color blindness.
4. 7.6 million (3.1%) have a hearing impairment. They might rely on transcripts and / or captions for audio and video media.”

This shows that the internet and websites have audiences from people both without and with some sort of disability. When websites or tool are poorly designed it could cause difficulties or even limit people with disabilities using them. Thus, developers need to incorporate the standards of web accessibility while designing their product.

As per the statistics from statscounter [28] of GlobalStats website, google chrome is the most popular browser in the recent years that web users prefer to use. 63.9% of all the users prefer to use chrome as their primary browser. Chrome integrated the voice capability into the browser by integrating the JavaScript Web speech api into the browser architecture from version 25 [29]. These make chrome to be a preference of choice for the VCAT tool to be built as a chrome browser extension.

The Voice Controlled Accessibility and Testing (VCAT) tool is primarily aimed to help users perform the following tasks:

1. Navigate the web using voice commands, and
2. Automatically generate Java-Selenium test cases.

VCAT is a browser extension that runs in the chrome browser. The tool waits for the user to speak into the microphone and provide commands and executes those commands on the currently opened web page. The following are requirements for VCAT to work properly:

1. A working microphone is connected to the computer.
2. After following the installation instructions below, the VCAT options page will automatically open every time the chrome browser is launched. Note - the VCAT options tab must always be kept open for VCAT to accept and execute voice commands. In case the options page is closed, then click on the VCAT icon next to the address bar on the browser and then click on options.

When the VCAT tool is installed for the first time, the browser requests permission to access your microphone. You must click "Allow", to let the tool access your microphone.

As noted, the VCAT tool executes the commands that are given by the user on the currently active tab.

When VCAT tool is installed for the first time, the browser requests permission for access for Microphone access. The user must click allow to let the tool get access to the microphone.

3.2 Installation steps

The VCAT extension is published on chrome web store and the extension can be installed on the chrome browser from the Web store. The extension can be accessed on <https://chrome.google.com/webstore/detail/vcat/gjbnjhimmkpplfccacmmkglojjgimkld>

Once the above page is opened on a chrome browser click on the “**Add to Chrome**” button to install it on the browser.

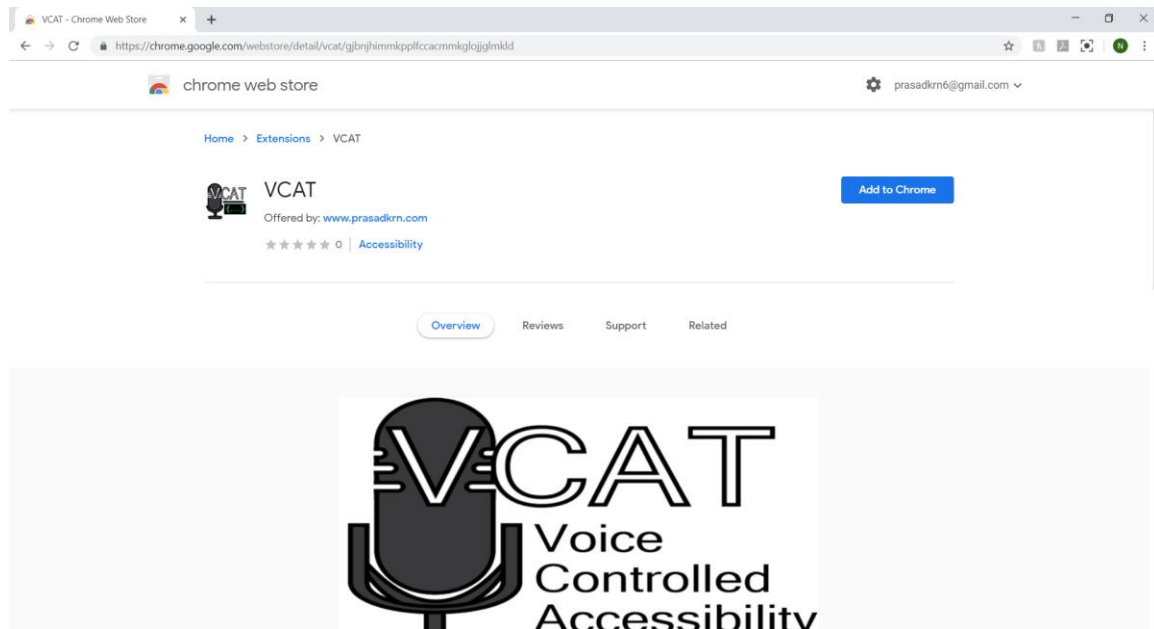


Figure 12 VCAT Extension on Chrome web store

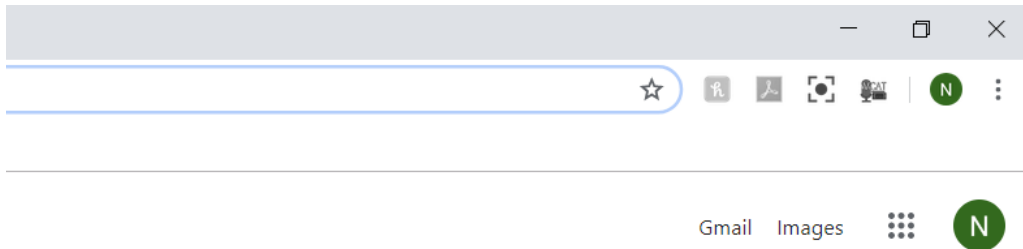


Figure 13 Chrome extension icon on the browser

For development purpose and for the study participation, VCAT was installed via the developer mode as described in the following steps:

1. Download the VCAT extension from the following link and extract the contents to a folder named "VCAT". Link to Download VCAT:
<https://github.com/prasadkrn83/VCAT>
2. Open the chrome browser.
3. Open the Extension Management page by navigating to `chrome://extensions`. The Extension Management page can also be opened by clicking on the Chrome menu, hovering over More Tools, then selecting Extensions.
4. Next, enable Developer Mode by clicking the toggle switch next to Developer mode.
5. Click the LOAD UNPACKED button and select the VCAT extension directory, where you extracted the VCAT.zip file. Make sure to select the VCAT directory name.

3.3 Application Design

The VCAT application is mainly divided into two main sections:

1. The Chrome browser extension module
2. The VCAT service module.

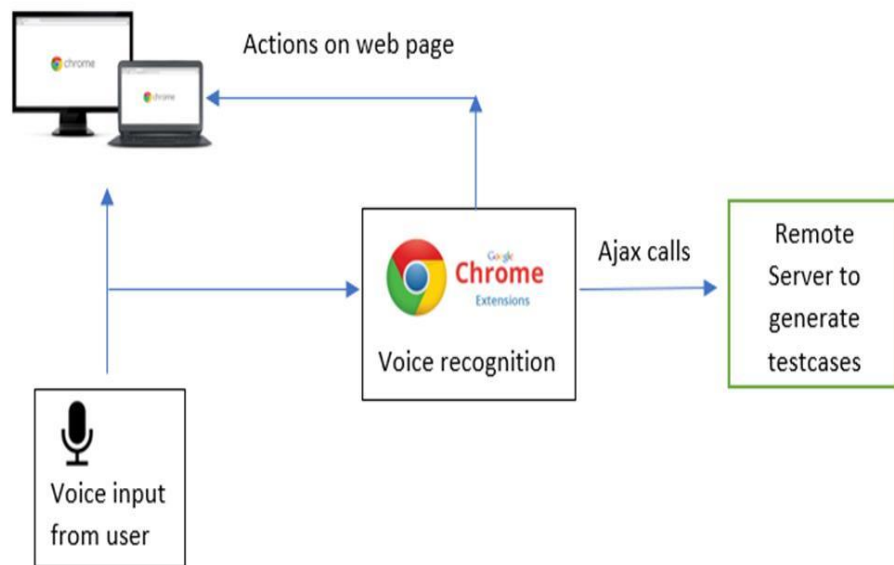


Figure 14 High level diagram of the VCAT Tool

3.4 The chrome browser extension module:

The chrome browser extension module is the component that is installed on the user's chrome browser. The extension would always be waiting for the user to give a voice command. Once the tool recognizes a voice input in the system microphone, it tries to interpret and match it to a given set of standard commands that are defined in the VCAT tool. If there is a match for a valid command and the current tab that is active on the browser is a valid webpage then the tool would try to execute the command on the webpage. Else it would just ignore the input voice command. The process flow of execution in the VCAT tool is illustrated in the following flow diagram:

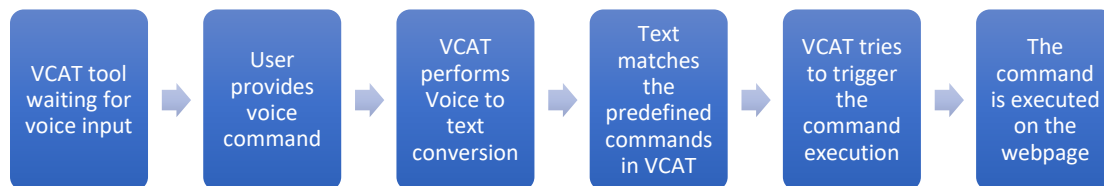


Figure 15 Flow diagram of VCAT

3.5 Components of the VCAT Extension

The VCAT Extension has the following components:

1. Option (vcat.html): This page is the option page of the extension. The option page serves the following functionality in the application:
 - i) ON/OFF: This is a toggle button used to turn on or off the VCAT tool from listening to voice commands from the user.

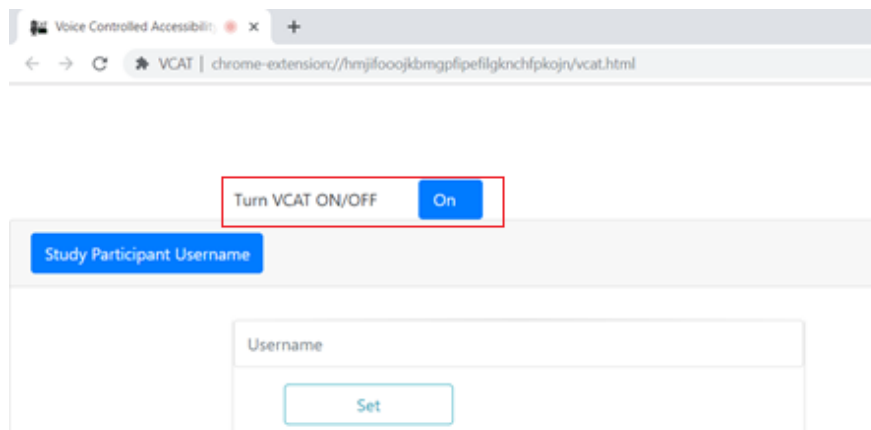


Figure 16 VCAT Configuration : Turn On/OFF

- ii) Auto complete configuration:

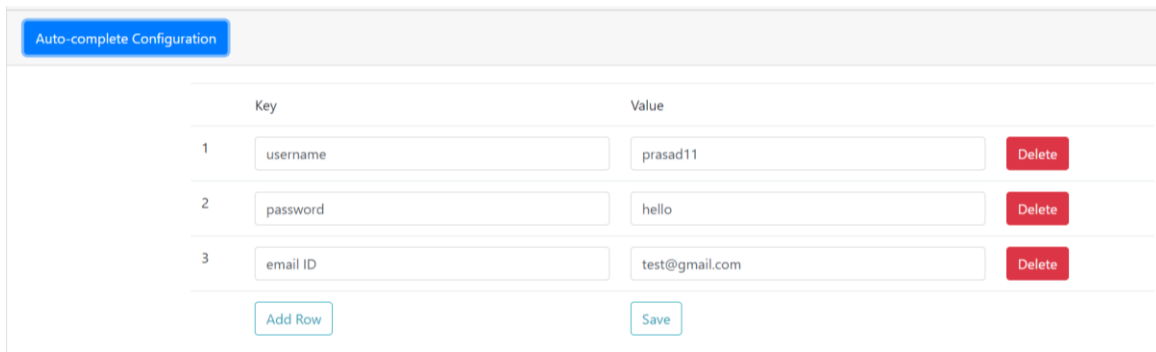
In situations where text values containing non-natural English language words, the Auto Complete Configuration section can be used as an alternative. The auto complete entries are identified by "key"- "value" pairs and can be added or removed per the user's needs. The "key" denotes the reference to the text and the "value" denotes its value. The Auto Complete Configuration section can

also be used to easily enter repetitive values, such as addresses, names, frequently used words.

Note: For the auto complete to work, the key must be a natural English word.

For example, assume a user has a username, such as "user_12name". The value "user_12name" is a non-natural English word and would not be recognized by the speech recognition component. Therefore, the auto complete configuration section can be used, to create a key-value pair, where the Key is the word "Username", and the value is "user_12name".

To reference this entry while completing a text field, the key must be referenced as hash . For example : Enter Value Hash Username. This command will set the selected text field value to "user_12name".



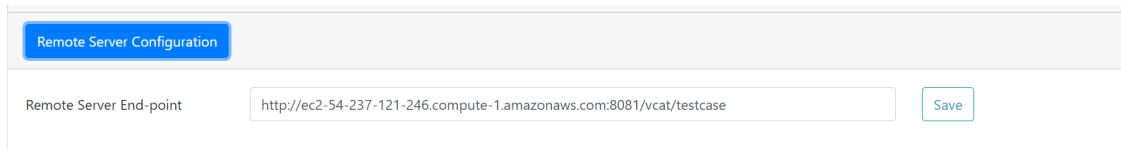
The screenshot shows a web interface titled "Auto-complete Configuration". It features a table with two columns: "Key" and "Value". There are three rows of data, each with a "Delete" button to its right. Below the table are two buttons: "Add Row" and "Save".

	Key	Value	
1	<input type="text" value="username"/>	<input type="text" value="prasad11"/>	<input type="button" value="Delete"/>
2	<input type="text" value="password"/>	<input type="text" value="hello"/>	<input type="button" value="Delete"/>
3	<input type="text" value="email ID"/>	<input type="text" value="test@gmail.com"/>	<input type="button" value="Delete"/>

Figure 17 VCAT Configuration: Autocomplete

iii) Remote server configuration:

This is used to configure the end point of the VCAT service which is hosted on a remote server. It is a restful webservice that will be used by the VCAT extension to generate the java selenium test case



Remote Server Configuration

Remote Server End-point Save

Figure 18 VCAT Configuration: Remote Server Endpoint

2. background script: The background script is responsible for the following functionalities in the extension
 - a) Initialize the speech recognition in the extension.
 - b) Perform speech to text conversion.
 - c) Recognize valid commands given by the user.
 - d) Send message to the content script to execute the command.
 - e) When the given speech command does not match the syntax of the standard command syntax, perform cosine similarity calculation. The command is then tried to be rephrased and rematch to the standard commands. If there is a match then the command is triggered else, ignored.
 - f) Send notifications to the content script to display messages.
 - g) Perform ajax call to the VCAT service to generate the selenium code.

3. content script: The content script is used to perform the actions that the user provided via the voice commands. The content script is responsible for the following functionalities:
 1. Extract the command from the message sent from the background page.
 2. Try to execute the command on the web page.
 3. Extract the xpath of the element if any was in the command that was executed.
 4. Display information toast message on the webpage.
 5. Display error toast message on the webpage if the command execution failed.
 6. Display the generated java selenium source code.

3.6 VCAT Tool Voice Commands:

VCAT processes the voice commands in the following sequence of events.

Step-1: On the launch of the VCAT options tool launches the options component and the background script.

Step-2: The Background script initializes the anyyang framework to accept voice commands.

Step-3: The VCAT tool waits indefinitely for the user to speak into the microphone. As soon as the user starts speaking, the anyyang framework goes into the listening mode and starts to interpret the speech using the chrome browsers built-in Speech Recognition interface of the Web Speech API.

Step-3: Anyyang tries to match the recognized text to the predefined commands.

Step-4: If there is a match then the background page constructs a message object with the command details and communicates it using the sendMessage() method to the currently active window.

Step-4: If there is no match then anyyangs “resultNoMatch” callback function is triggered.

Step-5: The callback function uses cosine similarity matching to the predefined array of keywords used in the commands to reconstruct the voice command text.

Step-6: The reconstructed strings are used to manually fire the trigger() command of anyyang.

Step-7: If there is a match from the new command, then the correct command is triggered and. Else the input voice command is ignored and the VCAT waits for new voice command to be given by the user.

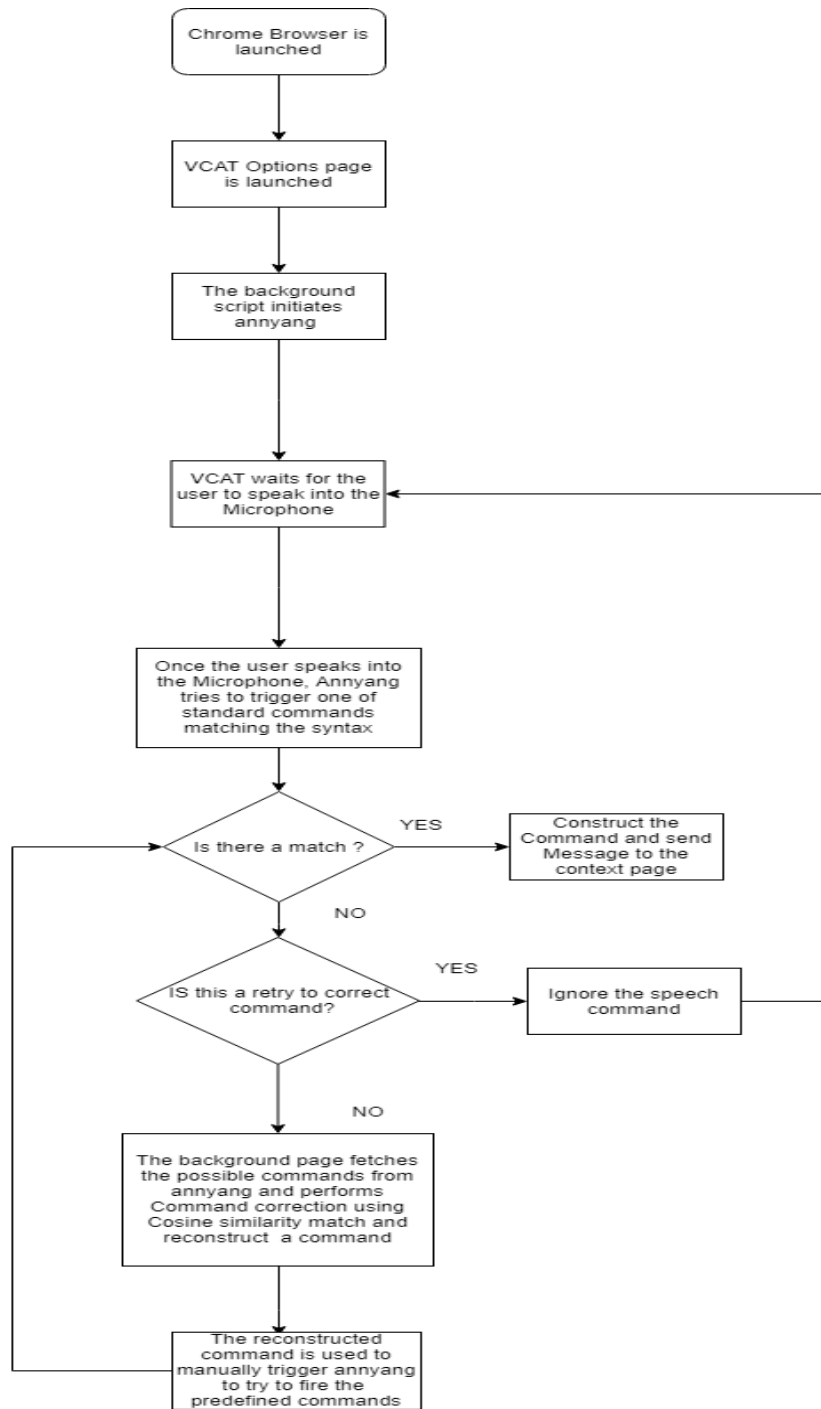


Figure 19 Flowchart of VCAT recognizing voice command

3.6.1 *VCAT Trigger word*

After prolonged inactivity in VCAT, the speech recognition would go into a sleep mode. A detection of any speech would activate VCAT again into listening mode. Sometimes it was observed that the first word was missed between the activation of VCAT into the listening mode and recognition of the spoken words. Hence to ensure that the complete command is recognized a hot word “computer” is used. Hence every command in VCAT should be prefixed by the hot word “Computer”. Skipping the hot word would not stop the VCAT from recognizing or executing the recognized command but the efficiency of success is low.

3.6.2 *VCAT voice commands*

The VCAT tool has number of voice commands that are broadly classified in the following seven groups:

1. Basic Browser operations.
2. Scrolling through web page.
3. Select Command.
4. Click Command.
5. Set/Enter Command.
6. Generate Test case command.
7. VCAT Configuration specific commands.

a) Basic Browser operations

The basic browser operations are used to operate the browser as a whole and not limited to the webpage within the tab.

The following commands are available to use:

Command Name	Description
Open New Tab	Used to launch a new, blank tab.
Open <url>	Used to launch the url provided in the command on the current tab. Example: open google dot com to launch "google.com".
Refresh Refresh Web Page	Used to refresh the current tab.
Go Back	Used to trigger the back button click on the current tab.
Go Forward	Used to trigger the forward button click on the current tab.
Close Tab	Used to close the current tab.

Table 3 VCAT Basic browser operational commands

b) Scrolling through the web page

The scroll commands are used to scroll the page in the specified direction. The following commands are available to use:

Command Name	Description
Scroll Page Up	Used to scroll the web page Up by one screen.
Scroll Page Down	Used to scroll the web page Down by one screen.
Scroll Page To Top	Used to scroll the web page to the Top.
Scroll Page To End	Used to scroll the web page to the bottom End.

Table 4 VCAT Scroll commands

c) Select Command

The select command is used to highlight a specific element on the web page. There are two types of possible select operations on the web page:

- a) Select all: This is used to select all the elements on the web page of a particular type. The following types of elements are recognized by VCAT:

Element on the Webpage	VCAT name
Link	Link
Text boxes	Text box
Buttons	Button
Check boxes	Check box
Combo box	Combo box

Table 5 Mapping of HTML elements in VCAT

Example usage:

Command: **Select all Links**

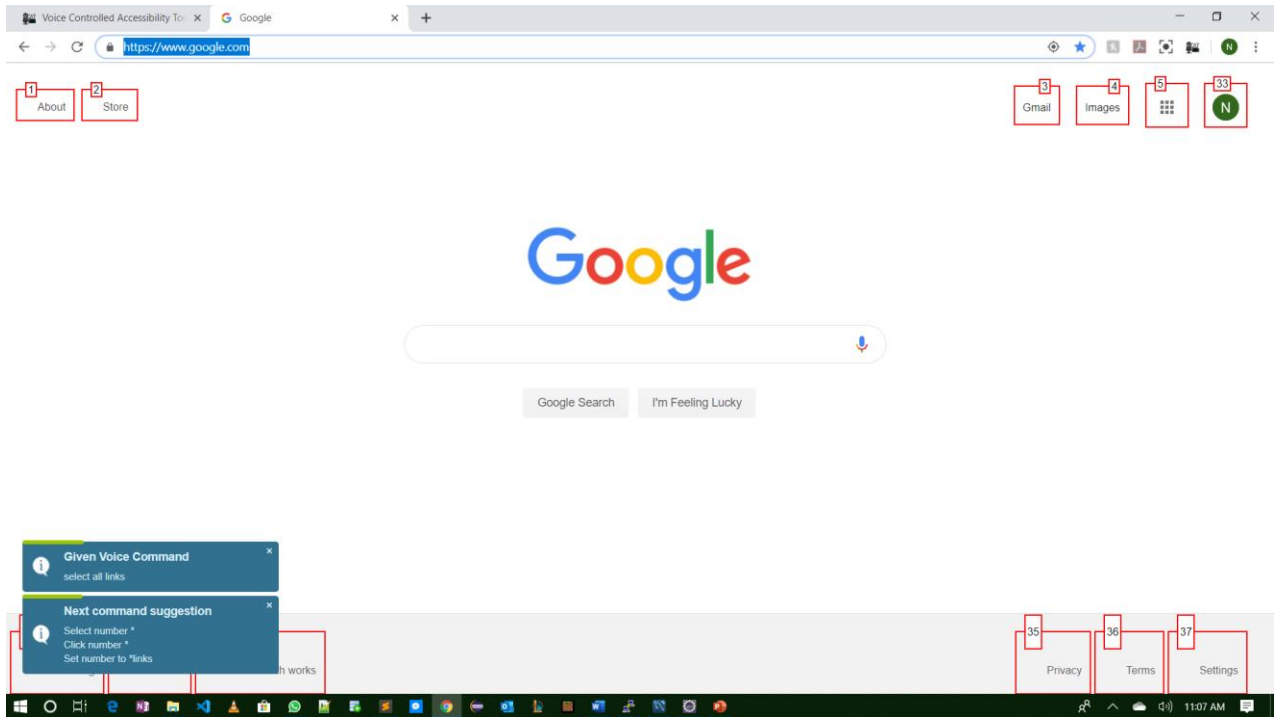


Figure 20 VCAT Select All Links command

Command: **Select all Buttons**

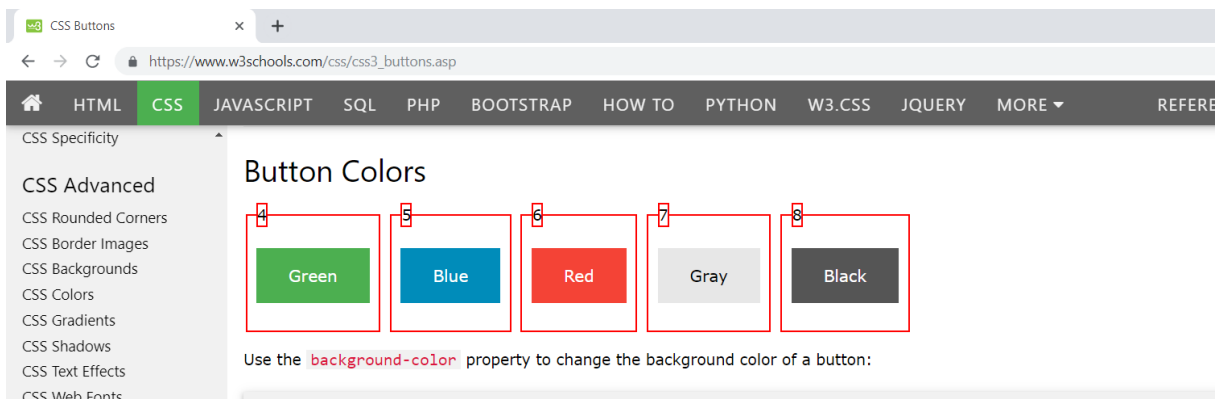


Figure 21 VCAT Select All Buttons Command

Command: Select all Text Boxes

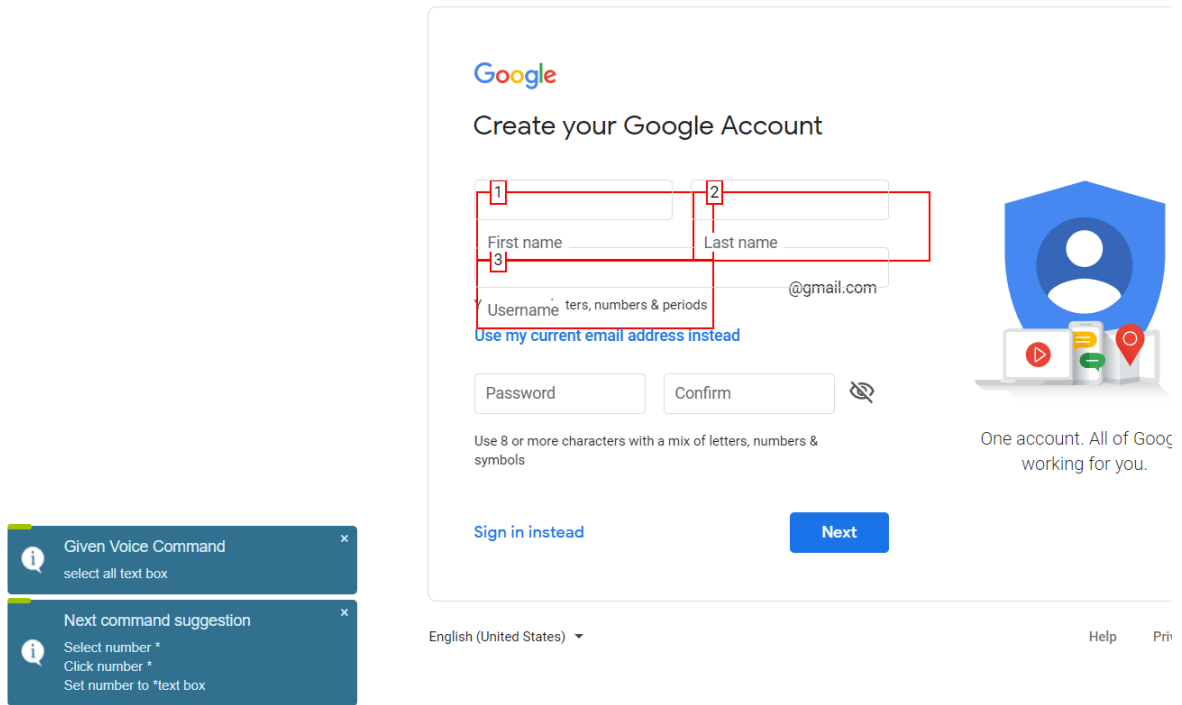


Figure 22 VCAT Select All Text Boxes Command

- b) **Select Number <number>'**: Once the elements have been selected, they will be numbered. This command is used to select an element on which the user wants to trigger further action. Example: Select number 1

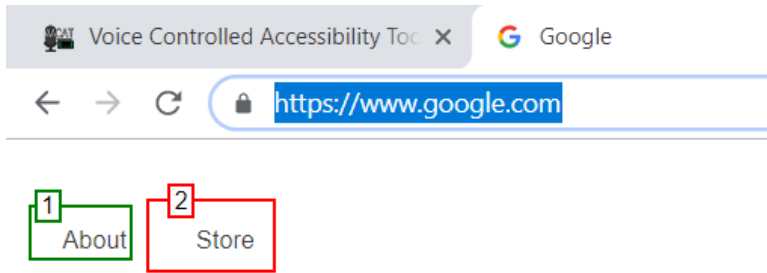


Figure 23 VCAT Select Number Command

- c) Deselect: Can be used to deselect all selected elements on the page. If there are any elements that are selected, then all the selections will be removed.

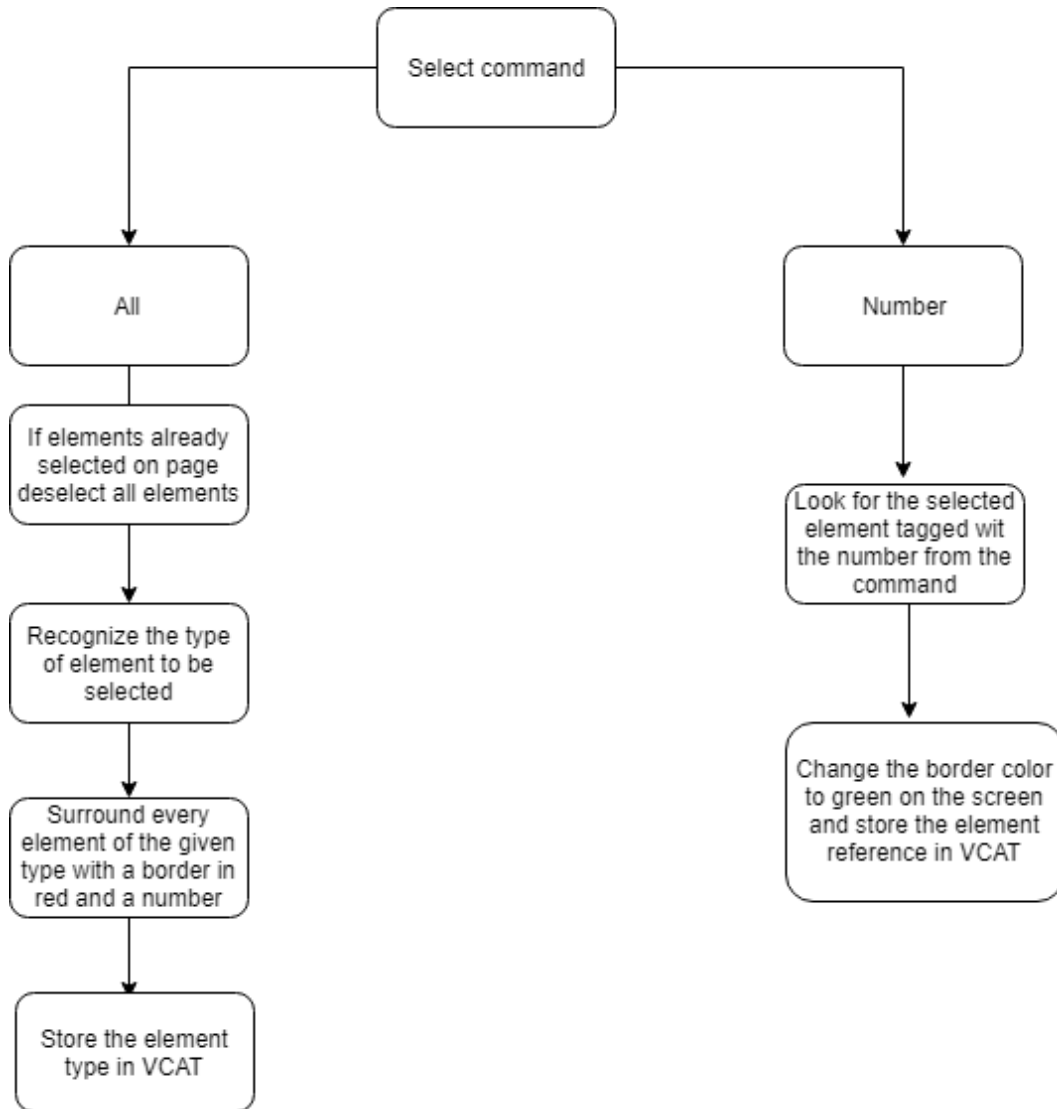


Figure 24 Flowchart for Select Command

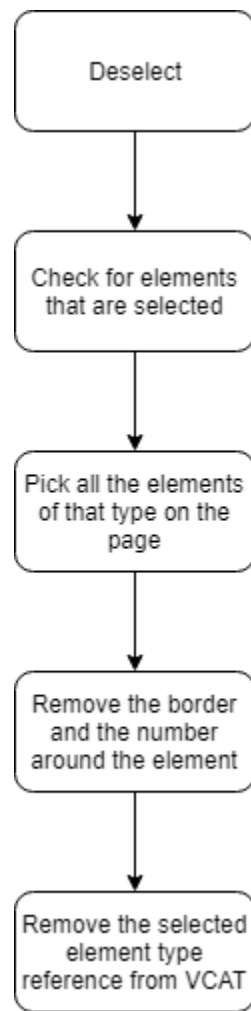


Figure 25 Flowchart for Deselect Command

d) Click Command

The click command is used to click specific element on the web page. There are two types of possible click operations on the web page:

- i. Click on number <number>: The command is used along with the select all command. Used to issue a click command to any selected element on the web page. Example: **Click on Number 5**.

Click on <label>: Used to click an element that can be identified by an associated label on the web page. Labels described by the aria tags can also be used in this command.

Example : **Click on Home**.

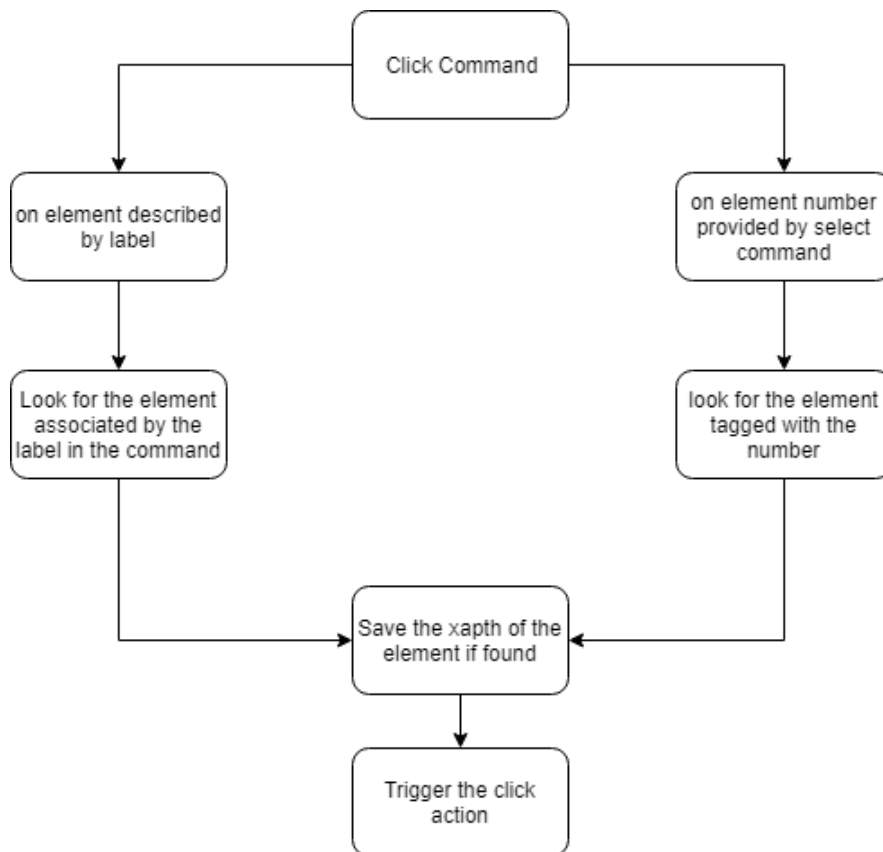


Figure 26 Flowchart for Click Command

e) Enter Command

This command is used to fill up text boxes on the web page and can be done in the following ways:

- i. Set As <value>: Used to set the value of the element identified by the label on the web page, to the given value. Example: Set Search As Java. Here, Search is the label of the text box, and Java is the value you want entered.
- ii. Enter Value <string>: Used to set the value of the selected text box to the given value. Example of steps to enter text into a text box:
 - a. Select All Text Boxes.
 - b. Select Number 5.
 - c. Enter Value “Java selenium”.
- iii. Enter Value Hash <keyname>: Used to set the value of the currently selected text box, to the value of the key defined in the VCAT configuration page. If the given key in the command is not found in the configuration page, then the command is ignored.

Example of steps to use Hash Values:

- a. Select All Text Boxes.
- b. Select Number 5.
- c. Enter Value Hash UserName (note - in this example, UserName was predefined as a key-value pair in the configurations page.)

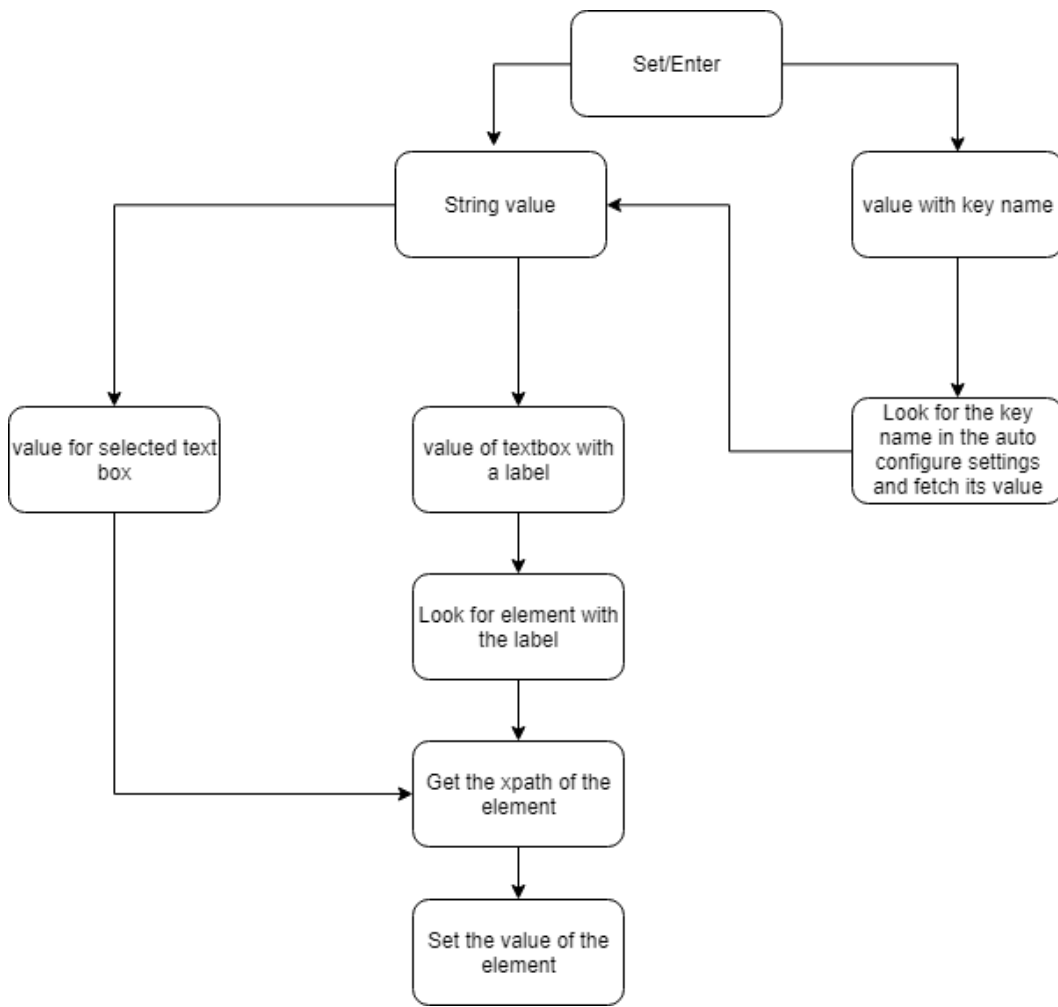


Figure 27 Flowchart for Set/Enter Value Command

f) Selenium Test Case Generation Commands

Generate Selenium Test Case A selenium test case will be generated with the sequence of commands instructing the VCAT as follows:

- a) **Generate Test Case:** This will turn on the test case generation mode of the tool.
All the successful voice commands will be recorded to be used in the test case generation.

- b) **Complete Test Case:** This will turn off the test case generation mode of the tool.
 - a. Now the VCAT tool will continue to accept voice commands and execute them on the web page, but no steps are recorded. The Commands that have been recorded up until stop the test case, will be submitted to the VCAT service.
 - b. Once the VCAT service generates the test case and responds back to the VCAT tool, the generated source code is displayed on the browser in an overlaying message dialog box.

- c) **Close Message:** Used to close the message dialog overlay from the previous step.


```
Generated Selenium Test Case

package com.org.vcat;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.By;
public class TestCase {
public static void main(String[] args) throws Exception{

    System.setProperty("webdriver.chrome.driver", "D:\\ChromeDriver\\chromedriver.exe");
    WebDriver driver = new ChromeDriver();
    driver.get("https://www.google.com/");
    Thread.sleep(1000);

    testCaseMethod2();

    Thread.sleep(10000);
    driver.close();
}
public static void testCaseMethod2(){

WebElement inputBox_48 = driver.findElement(By.xpath("//*[id="tsf"]/div[2]/div[1]/div[1]/div[1]/div[1]/input[1]"));
inputBox_48.sendKeys("Google");
Thread.sleep(100);
    WebElement submit_62 = driver.findElement(By.xpath("//*[id="tsf"]/div[2]/div[1]/div[2]/div[2]/div[2]/center[1]/input[1]"));
    submit_62.submit();
    Thread.sleep(1000);
    Thread.sleep(10000);
    driver.close();
}
}
```

Figure 28 Message dialog displaying the generated selenium test case on the browser

g) VCAT Configuration Page specific commands:

Command Name	Description
Add Row	Used to add a new row to the auto-complete configuration table.
Delete Row <number>	Used to delete the entry in the auto-complete configuration table identified by its row number.
Save	Used to save the changes made on the auto-complete configuration table.
Select Key Number<number>	Used to select the Key of the row in the auto-complete configuration table identified by the row number.
Select Value Number<number>	Used to select the value of the row in the auto-complete configuration table identified by the row number.
Expand Auto Complete Configuration	Used to expand the autocomplete section

Table 6 VCAT Configuration screen specific commands

3.6.3 *Refencing elements by Label Name:*

Web developer have the freedom of labeling web elements in multiple ways.

- 1) Lable : Use the Label tag is used with an web element to describe the web element.

The labels are need to be linked to the element using either “for”, “airal-labeledby’ or “aria-label” attribued. Without any of these, it would be impossible for assitive programs to link an element with the label. VCAT makes use of these attributes to identify the element when used by its label name in the commands.

- 2) Value: Some elements like links or buttons, the label is a part of the element itself.

VCAT searches for elements with the value of the reference in the command to search for the element.

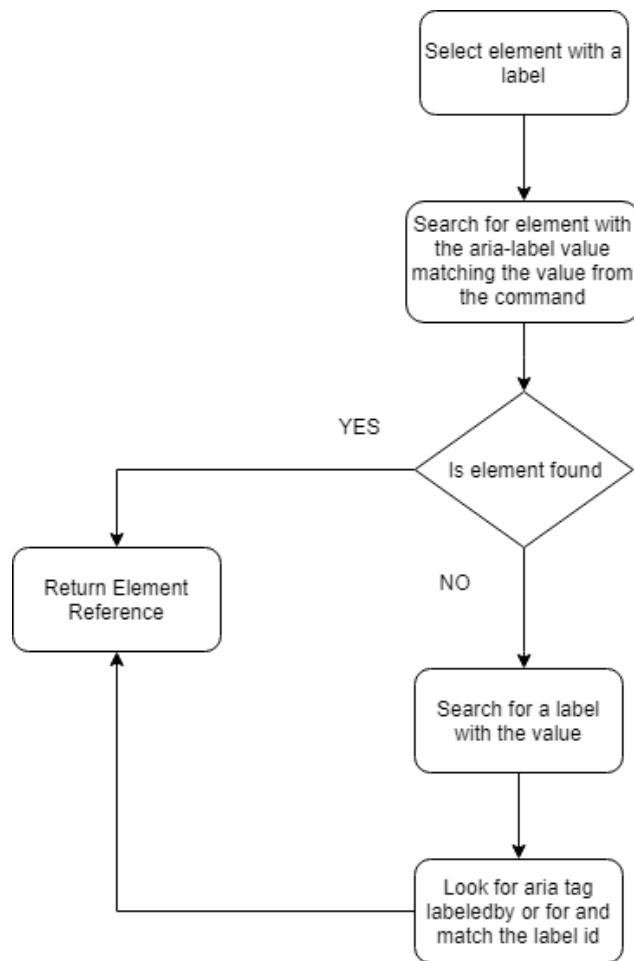


Figure 29 Flowchart for referencing element by label

3.7 VCAT Service module

3.7.1 Introduction

The VCAT Service module is used to generate the selenium test case from the steps captured by the VCAT chrome browser extension module. The service is implemented as a RESTful webservice on Spring boot framework. Once the user completes the sequence of steps on the browser and initiates the test case generation, the extension invokes the

webservice endpoint with the payload of the steps captured. The webservice iterates over the steps from the incoming request in the sequential order and generates the java selenium test case. The generated test case is currently inserted into the test case metrics table in the database.

3.7.2 *Service module design*

The VCAT service has the following components:

1. Controller: The Controller is responsible to handle all incoming requests and respond back to the client. The VCAT controller has two methods, one to produce a method to create the test case and the other to check for the username the subjects choose to participate in the study is not taken by any other subjects. The former method accepts POST requests and the latter a GET.
2. Code Generation: This component is responsible for the generation of the code, a factory pattern class to create instances of the web component based on the type of the component it is and finally the test case repository to persist the test case and the metrics in the database.
3. Beans: The bean classes represent the classes that are used to represent the state of the request, response, the web elements and finally the entity bean to represent the metrics table in the database.
4. Test case framework: This component represents the parts of a test case source code: the test script source code, import statements, methods, arguments and variables. All these classes are used to build the final test case source code.

5. Web Elements: The web elements are the different types of web elements that can be used in a selenium test case to simulate the actual elements of the webpage.

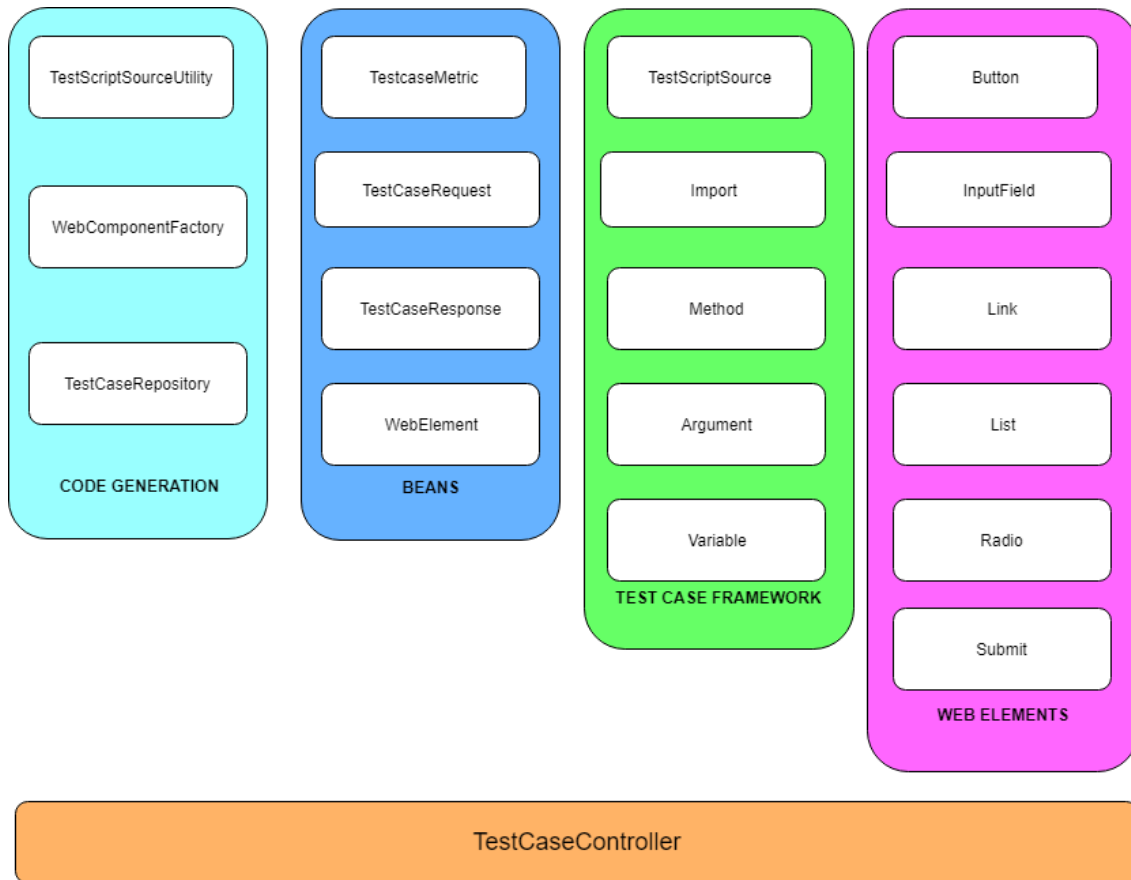


Figure 30 Components of the VCAT Service module

3.7.3 Steps to create a selenium test case source code

Step-1: The controller receives the request from the client call.

Step-2: The controller creates an instance of the Testcase Source.

Step-3: The source code is associated with the required import statement.

Step-4: Initialization code, the driver configurations code is added.

Step-5: A main method is created in the source code.

Step-6: A test method is created and added to the source code.

Step-7: The list of elements in the request is iterated over to create sequence of web elements and added to the test method.

Step-8: The test method is added to the source code.

Step-9: The reference to invoke the test method from the main method is added.

Step-10: The string value of the final test case source instance is added to the response object in the controller.

Step-11: Metric details such as operating system, version of the chrome browser from the client, time to create the teste case request, study participant name from the client and the actual test case are all stored into the test metrics table in the database.

Step-12: The controller sends the response to the client.

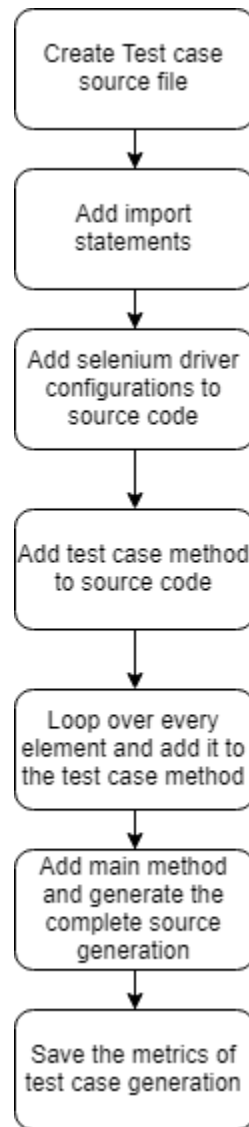


Figure 31 Flowchart for VCAT Service

4 Study Participation

4.1 Study participation process.

Since the study for the research involved human subjects, we submitted a protocol (IRB Protocol 2019-0278) to the Institutional Review Board (IRB) for approve. IRB reviewed the protocol and approved it as a “Minimal Risk”.

Post approval of the protocol students of University of Texas at Arlington were reached out with the survey consent form and pre study participation survey. A total of 20 participants took up the participation survey, out of which 17 students agreed to participate in the study and 3 students declined. Those students who agreed to participate were individually contacted to guide them the steps involved in the study participation. No personal information regarding the identity of the participants were captured as a part of the study.

4.2 Test cases for the Study participations

4.2.1 *Setting-up the Auto Complete Configuration Section.*

1. Create an auto complete configuration entry as "username"="username provided to you".
2. Create an auto complete configuration entry as "email"="test@gmail.com".
3. Create an auto complete configuration entry as "password"="password of your choice". (Note - please do not use a personal password, since this information is stored for using VCAT to test a login page, as part of participation study. Therefore, please use a generic, non-personal password.)

4.2.2 Testcase 1: Registration and login

A simple Registration and login test case. The demo of the execution of this test case is available on this link: Test case 1 demo

i) With the VCAT options tab open, open a new tab in the chrome browser. b.

Open the test registration page, in the new tab:

<http://www.prasadkrn.com/register.php>

Voice command sequence for registration:

- 1) Computer Generate Test Case
- 2) Computer Set Username as Hash Username
- 3) Computer Set Email as Hash Email
- 4) Computer Set Password as Hash Password
- 5) Computer Set Confirm Password as Hash Password
- 6) Computer Click on Submit
- 7) Computer Complete Test Case

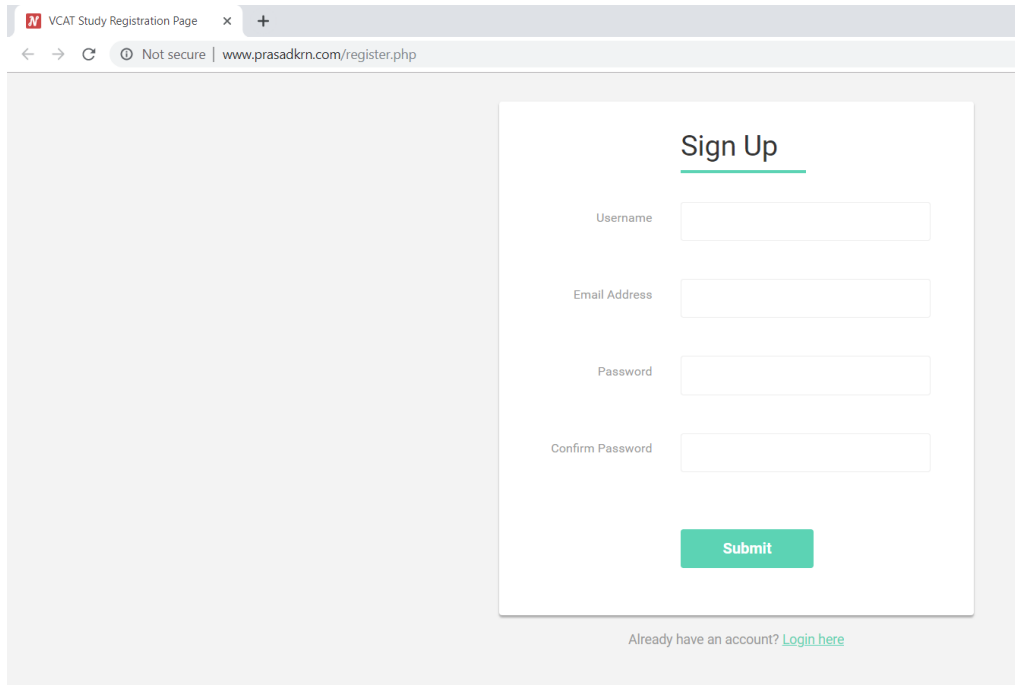


Figure 32 Testcase 1: Registration Page

Voice command sequence for login:

After completing the test registration portion above, use these voice commands:

- 8) Computer Generate Test Case
- 9) Computer Set Username as Hash Username
- 10) Computer Set Password as Hash Password
- 11) Computer Click on Login
- 12) Computer Complete Test Case.

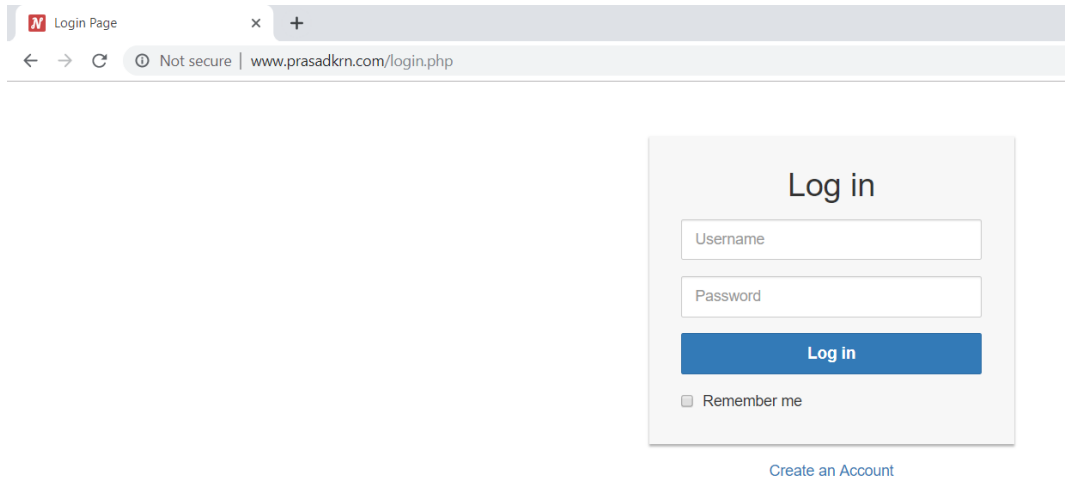


Figure 33 Testcase 1: Login page

4.2.3 Test case 2: Google search

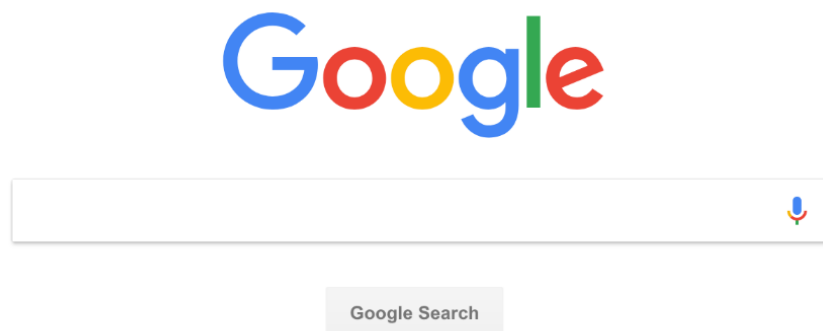


Figure 34 Testcase 2: Google Search

Give the following voice commands in the order of sequence mentioned below:

- 1) Computer Open New Tab
- 2) Computer Open Google Dot Com
- 3) Computer Generate Test Case
- 4) Computer Set Search as Java

5) Computer Click on Google Search

6) Computer Complete Test Case.

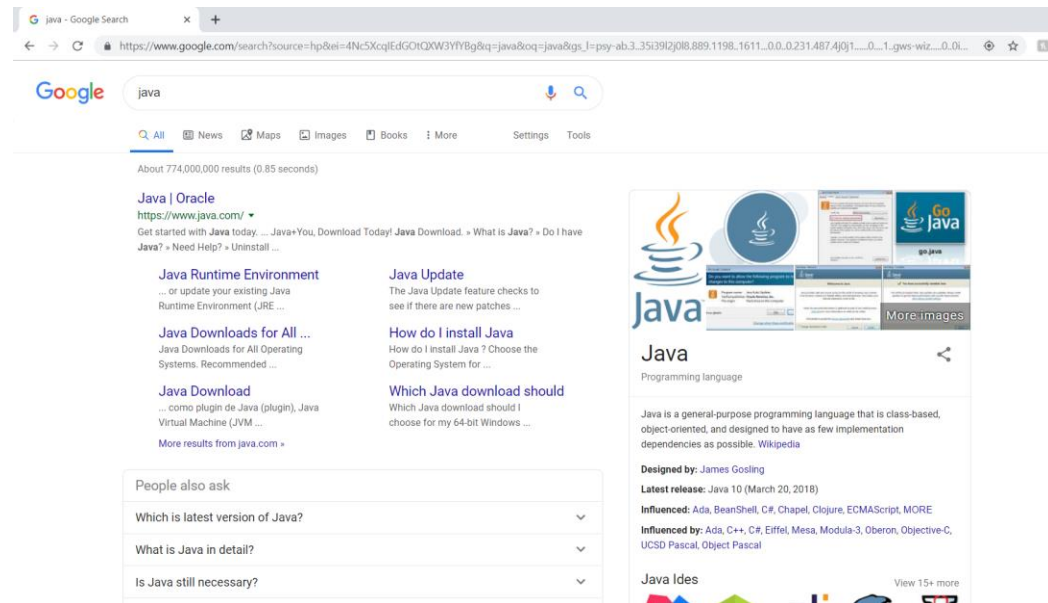


Figure 35 Testcase 2: Google search result

4.2.4 Test case 3: Expedia Flight Search

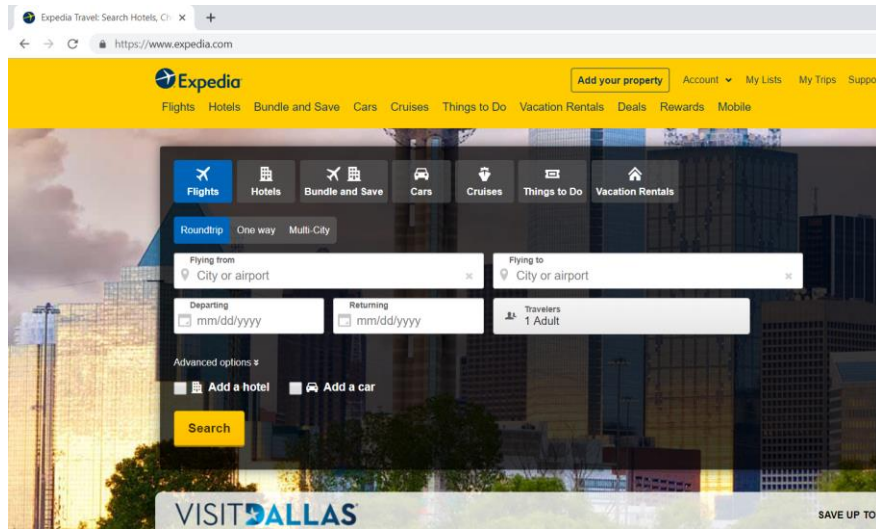


Figure 36 Testcase 3: Expedia flight search

Give the following voice commands in the order of sequence mentioned below:

- 1) Computer Open New Tab
- 2) Computer Open Expedia Dot Com
- 3) Computer Generate Test Case
- 4) Computer Select all Text Boxes
- 5) Computer Select Number <#> _(Note - the # corresponding to the Origin field.) _
- 6) Computer Enter Value Dallas
- 7) Computer Select Number <#> _(Note - the # corresponding to the Destination field.) _
- 8) Computer Enter Value New York

- 9) Computer Select Number <#> _(Note - the # corresponding to the Departure date field.) _
- 10) Computer Enter value 10 August 2019
- 11) Computer Select number <#> _(Note - the # corresponding to the Return date field.) _
- 12) Computer Enter value 20 August 2019
- 13) Computer Select All Buttons
- 14) Computer Click Number <#> _(Note - the # corresponding to the Flight Search button.)
- 15) Computer Complete Test Case

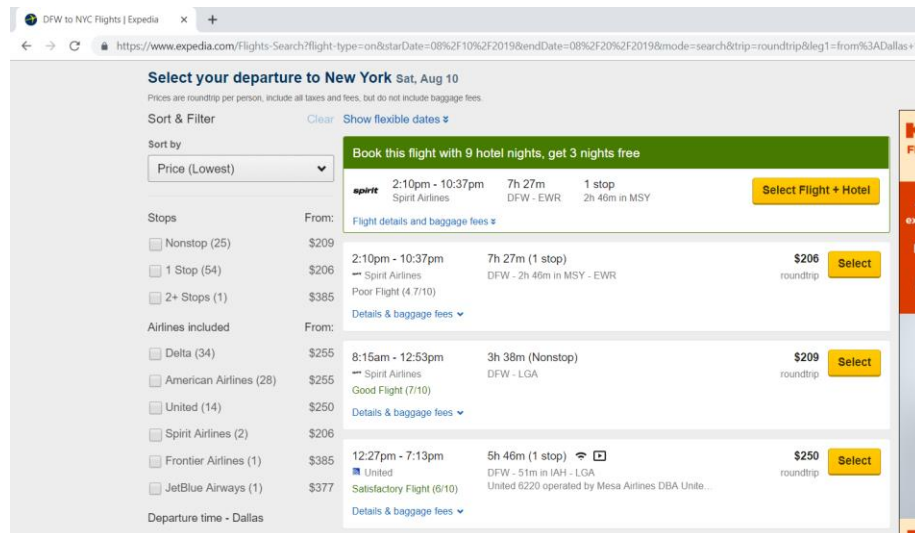


Figure 37 Testcase 3: Results on Expedia from flight search

4.2.5 Test case 4: Amazon shopping cart

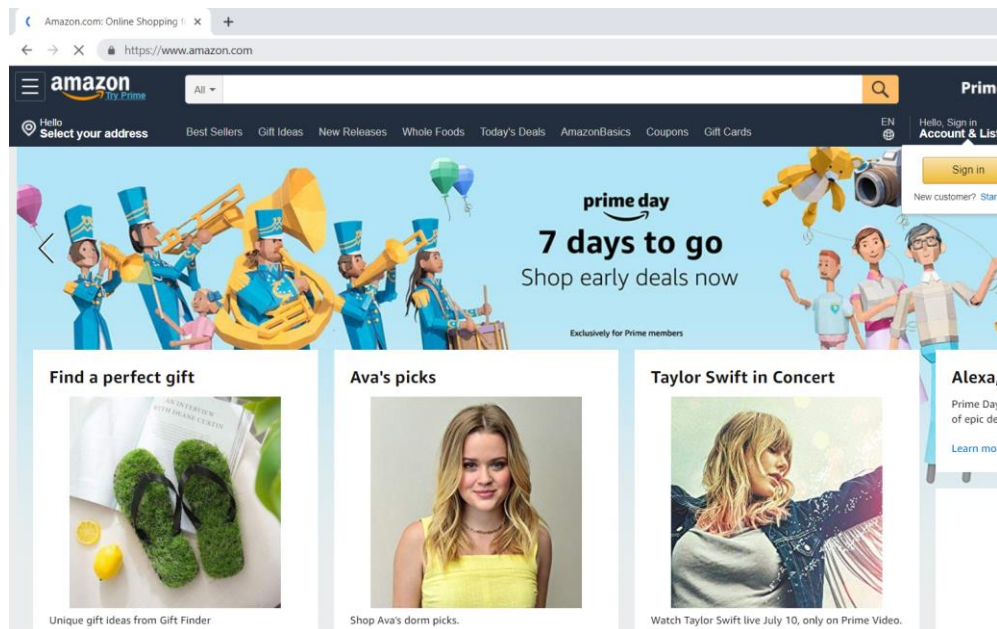


Figure 38 Testcase 4: Amazon Shopping cart

Give the following voice commands in the order of sequence mentioned below:

- 1) Computer Open New Tab
- 2) Computer Open Amazon Dot Com
- 3) Computer Generate Test Case
- 4) Computer Set Search as Paint Brushes
- 5) Computer Submit
- 6) Computer Scroll Page Down
- 7) Computer Select All Links
- 8) Computer Click Number <#> _(Note - any # corresponding to any image of Paint Brushes)
- 9) Computer Click on Add To Cart

10) Computer Complete Test Case

4.2.6 Test case 5: UTA Website

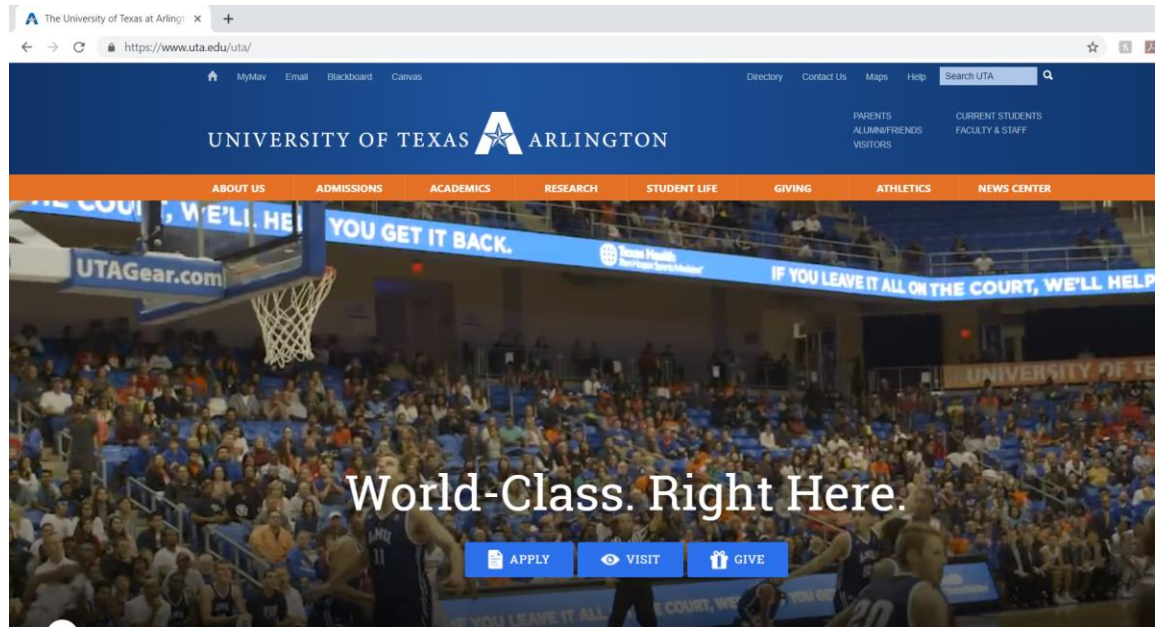


Figure 39 Testcase 5: UTA website

Give the following voice commands in the order of sequence mentioned below:

- 1) Computer Open New Tab
- 2) Computer Open UTA Dot Edu
- 3) Computer Generate Test Case
- 4) Computer Scroll Page to End
- 5) Computer Click on Admissions
- 6) Computer Scroll Page to End

7) Computer Click on Apply Today

8) Computer Complete Test Case

5 Experimental Results and Analysis

5.1 Preparticipation Survey Metrics

Institutional Review Board (IRB) was approached for approval to get human study participants for evaluating VCAT : Protocol- 2019-0278.

A study participation consent survey was used to recruit UTA students to participate in the experiment.

A total of 20 students took up the survey; 17 students agreed to participate, and 3 students declined.

The participants were given with set of instructions how to set up VCAT on their system and execute 5 identified test cases.

Execution time and other details were captured from the participation to analyze the result.

Q1 – Consent to participate in the study

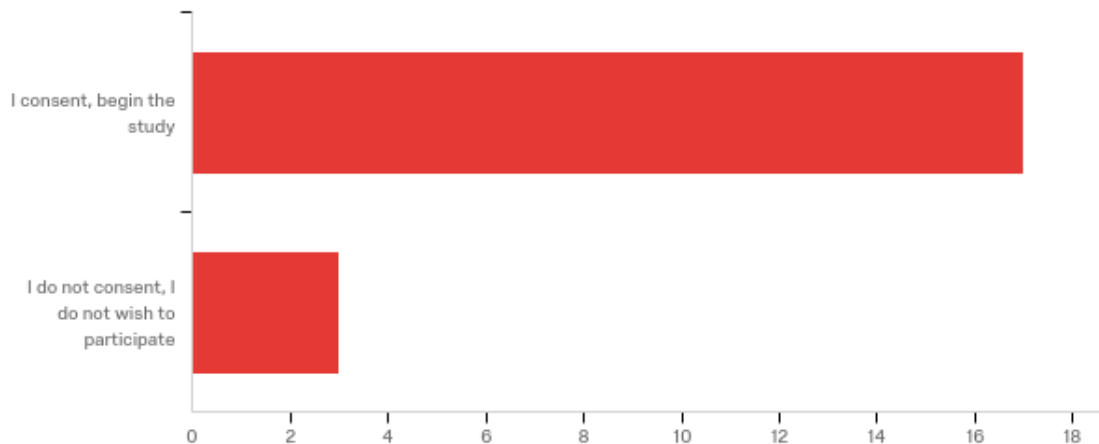


Figure 40 Study participant consent

Q2 - Gender

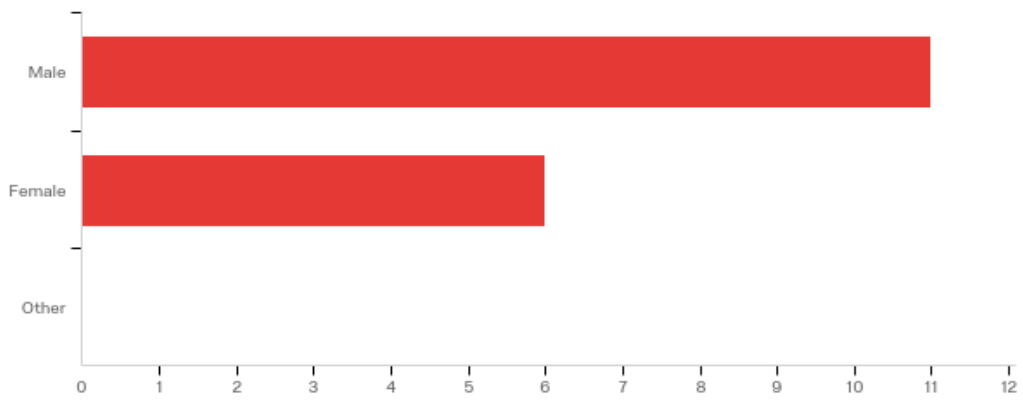


Figure 41 Study participant gender distribution

Q3 - Age

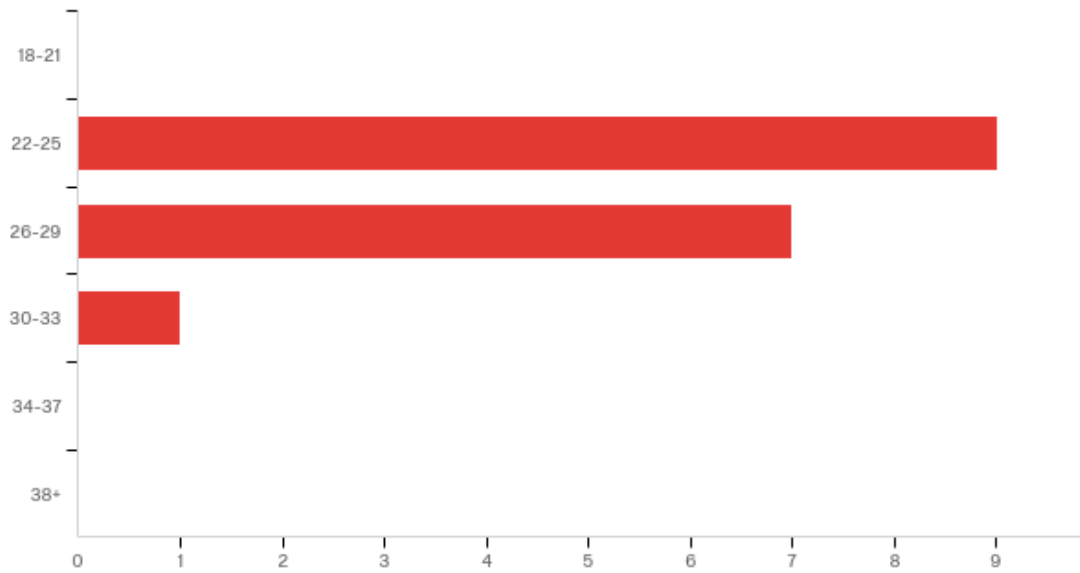


Figure 42 Study participant age distribution

Q4 - Highest degree received

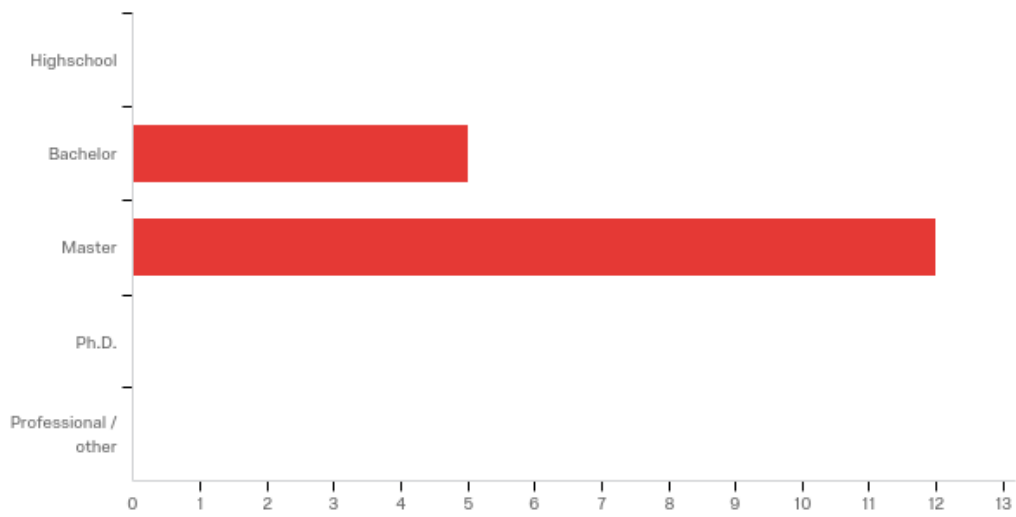


Figure 43 Study participants highest degree received distribution

Q5 - Current enrollment at UTA

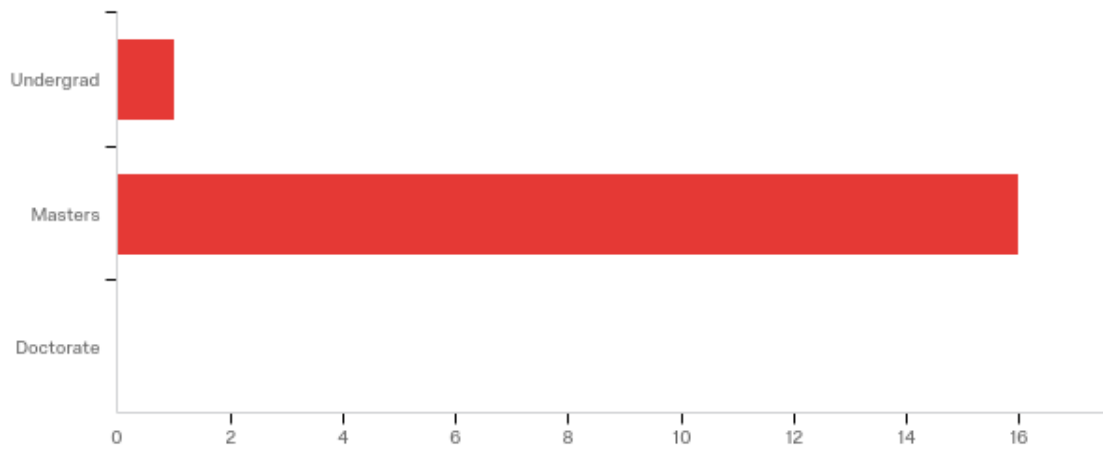


Figure 44 Study participants current UTA enrollment distribution

Q6 - Experience with using the Chrome web browser

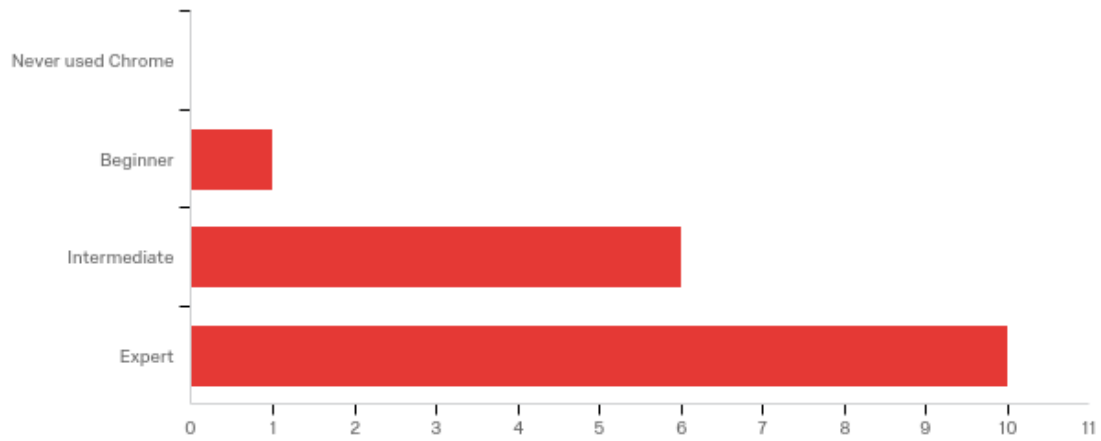


Figure 45 Study participants chrome browser usage experience distribution

Q7 - Do you have coding experience?

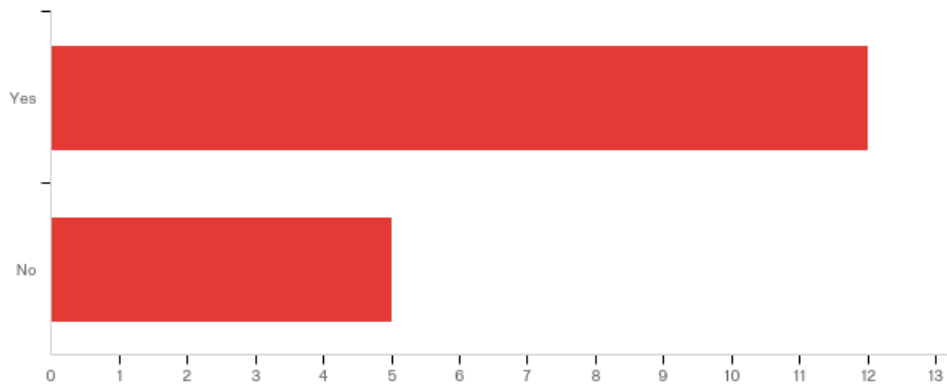


Figure 46 Study participants coding experience distribution

Q8 - How many lines of code have you written?

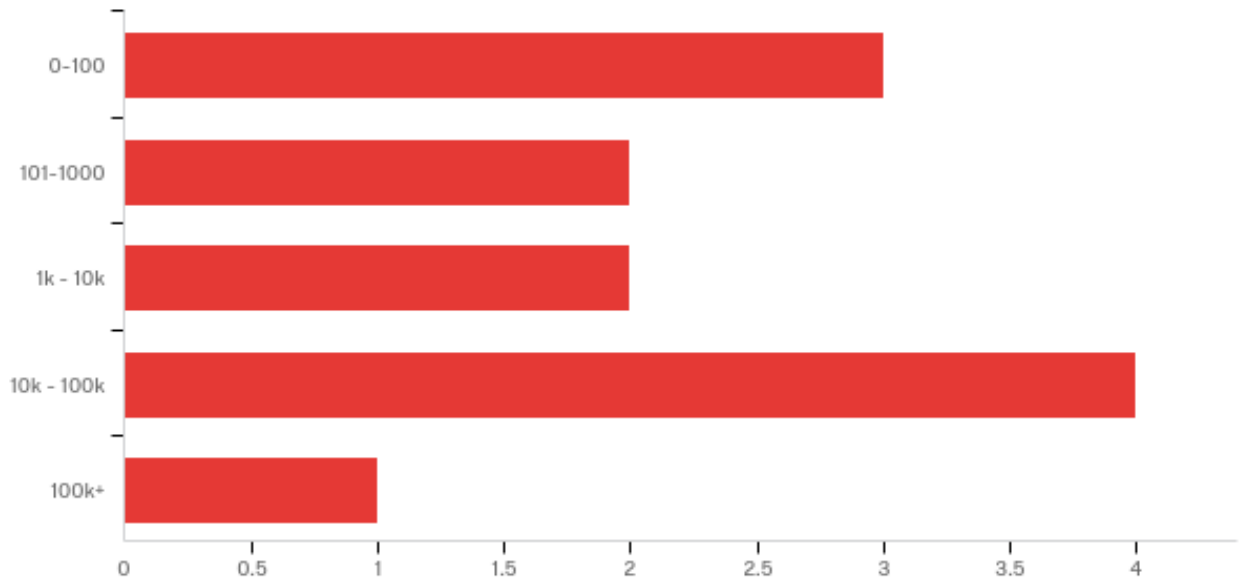


Figure 47 Study participants number of lines of written code distribution

Q9 - Have you written test cases (Selenium or other)?

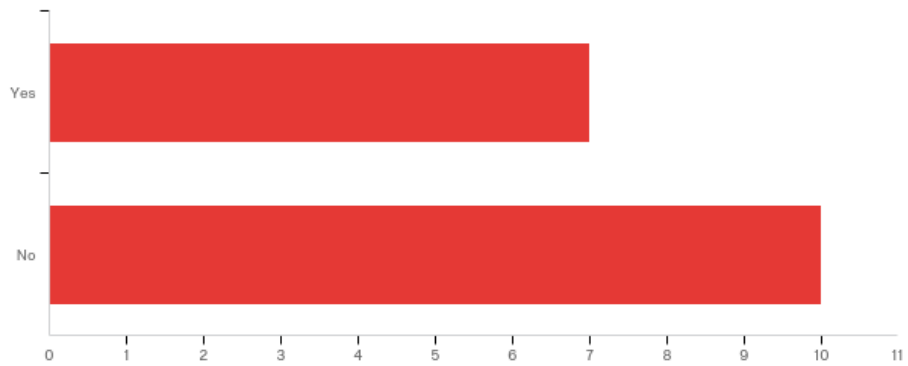


Figure 48 Number of Study participants who have written test cases

Q10 - Experience with writing Selenium test cases

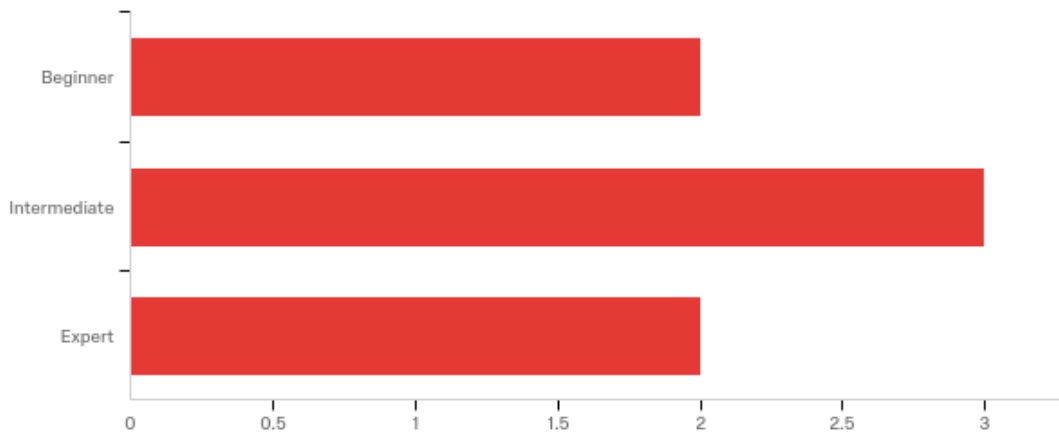


Figure 49 Study participants selenium experience distribution

Q11 - How many years of experience do you have writing Selenium test cases?

How many years of experience do you have writing Selenium test cases?	
Less than 1	
2	
0-1 years	
7months	
2	
1	
1	

Q12 - How many test cases have you written (Selenium or other)?

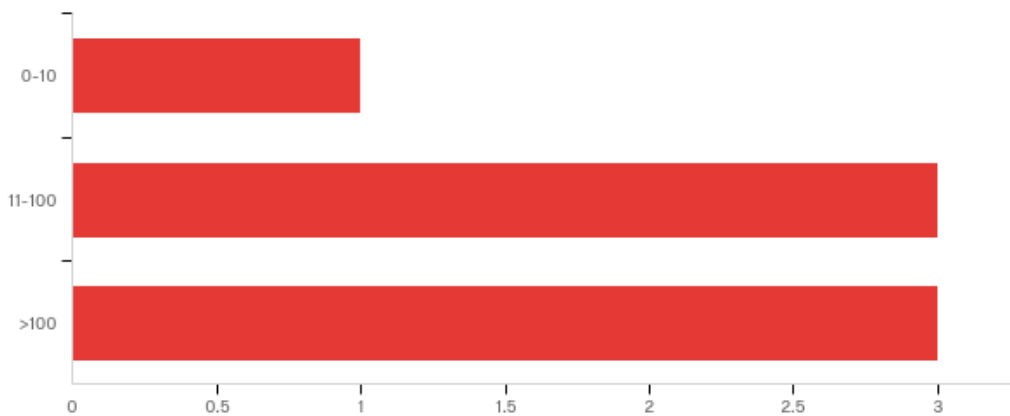


Figure 50 Number of test cases written by study participants

Q13 - If your answer to the previous "Selenium experience" question is Beginner/Intermediate/Expert, please review the test scenarios to be executed as a part of this study and estimate how many minutes you would need to write each test case manually in Selenium.

#	Field	User1	User2	User3	User4	User5	User6	User7
1	Scenario 1 (Register and login)	8	10	14	0	16	5	15
2	Scenario 2 (Google search)	10	9	10	0	17	4	10

3	Scenario 3 (Flight search)	15	10	16	0	18	6	17
4	Scenario 4 (Add item to Amazon cart)	11	12	18	0	25	5	23
5	Scenario 5 (Navigate UTA web site)	14	12	12	0	23	5	16

Table 8 Study participants test case estimation

5.2 Result Analysis

The study group executed the identified test cases and the results were monitored and tabulated as below:

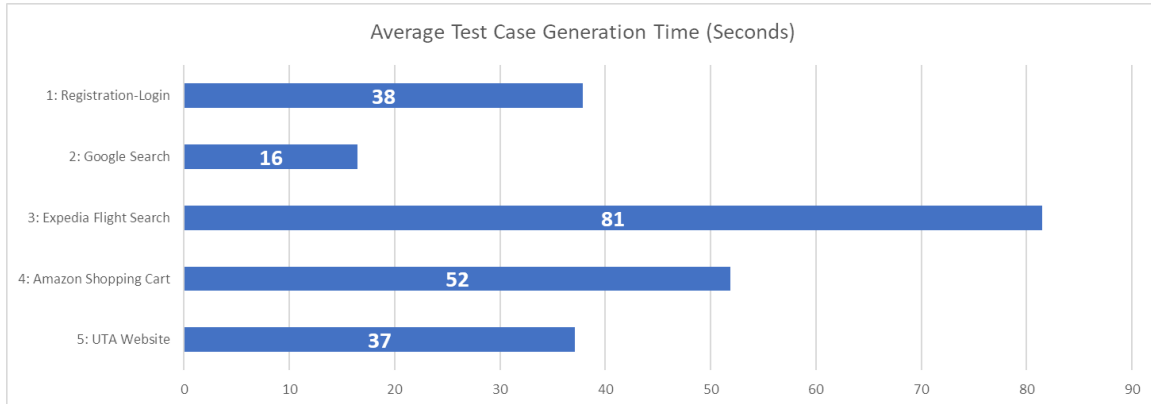


Figure 51 Average Time for test case generation in Seconds

5.3 Study participants results

User	Selenium Experience	Expected(sec)	Actual(sec)
User 1	Yes	480	41
User 2	Yes	600	45
User 3	Yes	840	47
User 4	Yes	0	36
User 5	Yes	960	44
User 6	Yes	300	27
User 7	Yes	900	43
User 8	No	0	30
User 9	No	0	29
User 10	No	0	40
User 11	No	0	25
User 12	No	0	33
User 13	No	0	45
User 14	No	0	33
User 15	No	0	54
User 16	No	0	39
User 17	No	0	33

Table 7 Test Case 1: Registration and Login

User	Selenium Experience	Expected(sec)	Actual(sec)
User 1	Yes	600	19
User 2	Yes	540	16
User 3	Yes	600	15
User 4	Yes	0	15
User 5	Yes	1020	18
User 6	Yes	240	15
User 7	Yes	600	14
User 8	No	0	14
User 9	No	0	23
User 10	No	0	13
User 11	No	0	19
User 12	No	0	13
User 13	No	0	16
User 14	No	0	15
User 15	No	0	21
User 16	No	0	13
User 17	No	0	21

Table 8 Test Case 2: Google Search

User	Selenium Experience	Expected(sec)	Actual(sec)
User 1	Yes	900	88
User 2	Yes	600	80
User 3	Yes	960	81
User 4	Yes	0	98
User 5	Yes	1080	102
User 6	Yes	360	75
User 7	Yes	1020	66
User 8	No	0	85
User 9	No	0	97
User 10	No	0	76
User 11	No	0	72
User 12	No	0	71
User 13	No	0	80
User 14	No	0	85
User 15	No	0	68
User 16	No	0	83
User 17	No	0	78

Table 9 Test Case 3: Expedia Flight Search

User	Selenium Experience	Expected(sec)	Actual(sec)
User 1	Yes	660	57
User 2	Yes	720	70
User 3	Yes	1080	46
User 4	Yes	0	57
User 5	Yes	1500	80
User 6	Yes	300	36
User 7	Yes	1380	45
User 8	No	0	34
User 9	No	0	51
User 10	No	0	56
User 11	No	0	36
User 12	No	0	33
User 13	No	0	38
User 14	No	0	65
User 15	No	0	75
User 16	No	0	52
User 17	No	0	51

Table 10 Test Case 4: Amazon Shopping Cart

User	Selenium Experience	Expected(sec)	Actual(sec)
User 1	Yes	840	32
User 2	Yes	720	40
User 3	Yes	720	42
User 4	Yes	0	41
User 5	Yes	1380	43
User 6	Yes	300	33
User 7	Yes	960	39
User 8	No	0	24
User 9	No	0	44
User 10	No	0	43
User 11	No	0	34
User 12	No	0	28
User 13	No	0	28
User 14	No	0	26
User 15	No	0	44
User 16	No	0	45
User 17	No	0	45

Table 11 Test Case 5: UTA website navigation

5.3.1 *Results*

1. It was observed that there was an increase in the accessibility of the browser with the use of VCAT tool.
2. There was a reduction in time over 85% for generating the Java Selenium Code when compared to the time estimate provided by the study participants.

5.3.2 *Specifications of the Server used in the Study participation:*

The VCAT server was an Amazon EC2 instance with the following specifications:

1. CPU:
 - product: Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
 - vendor: Intel Corp.
 - size: 2400MHz
 - capacity: 2400MHz
 - width: 64 bits
2. Operating System:
 - Linux #45-Ubuntu x86_64 GNU/Linux

6 Summary and Conclusions

6.1 Conclusion

To address the accessibility shortcomings of current websites, this work leveraged recent speech recognition advances to create a browser extension that interprets voice inputs as web browser commands and as steps in a corresponding test case. In this experimental study involving 17 participants, the resulting Voice Controlled Accessibility and Testing tool (VCAT) yielded a lower overall runtime than a traditional Selenium test case creation approach and improved accessibility for websites. Considering the positive results of this research project, the VCAT tool has tremendous potential as an extensible Chrome extension to assist both those with accessibility issues, and developers requiring faster test case development solutions.

6.2 Future enhancements

Several possible future enhancements have been identified and listed as below:

1. Improving method of displaying the element number: Currently the “select all” command uses the HTML feature of a fieldset and legend to add a number next to the element. This results in the change in the DOM structure of the page and hence the display and alignment of the page is altered. An improvement is to add numbering of the element without changing the alignment.
2. Language support: Chrome browsers Web speech api supports several languages across the world. The annyang api used in the research uses the chrome browser framework for speech to text conversion, the VCAT tool can be enhanced to work with other languages.

3. Generating complex selenium code: The code that is currently generated by the VCAT tool is basic. Testers in real world applications would need to write more complex test cases. The VCAT service and be enhanced to fully utilize the selenium capabilities and generate complex test cases automatically.
4. Customizing trigger word: Currently the VCAT extension is configured to use “Computer” as a trigger word. VCAT can be enhanced to provide option to the user to configure the trigger word of his/her choice.
5. Customizing the command syntax: The syntax of the voice commands defined in the tool. VCAT can be enhanced to provide the option to the user to define the command syntax.

Appendix A Operations that can be performed from selenium on the web elements

The following list summarizes the type of webelement and its corresponding actions that can be performed from selenium

1. Operations on edit box
 1. Enter a value
 2. Get the value
 3. Clear the value
2. Operations on link
 1. Click the link
 2. Return the link name
3. Operations on button
 1. Click enabled
 2. Display status
4. Operations on image
 1. General image(No functionality)
 2. Image button(click)
 3. Image link(redirects to another page)
5. Operations on text area

1. Return or capture the messages from the web page

6. Operations on checkbox

1. Select the checkbox
2. Unselect the checkbox

7. Operations on radio button

1. Select radio button
2. Check if it displays the radio button

8. Operations on drop down

1. Select an item in the list
2. Get the item count

9. Operations on frame

1. Switch from top window to a particular frame on the web page
2. Switch to a frame to top window

Appendix B List of libraries used:

1. Annyang : JavaScript based open source speech recognition library used to convert speech to text. <https://github.com/TalAter/annyang>
2. jQuery : JavaScript based open source library to perform HTML document traversal, modification etc. <https://api.jquery.com/>
3. Fuzzysset: JavaScript based open source library to perform lemmatization <https://glench.github.io/fuzzysset.js/>
4. Stopwordsremoval : JavaScript based open source library to perform stop words removal <http://geeklad.com>
5. Chrome extension api: Chrome api to develop extensions for chrome browser https://developers.chrome.com/extensions/api_index
6. Bootstrap : Opensource library to build responsive html pages <https://getbootstrap.com/>
7. Popper: JavaScript based open source library to add pop up functionality <https://popper.js.org/>
8. Spring boot: Open source platform to create java based standalone applications <https://spring.io/projects/spring-boot>
9. Java : Opensource programming language <https://www.oracle.com/java/>

Appendix C Links to VCAT

1. GitHub:

- a. VCAT Extension : <https://github.com/prasadkrn83/VCAT>
- b. VCAT Service Code : <https://github.com/prasadkrn83/VCAT-Service>

2. VCAT Demo Videos:

<https://www.youtube.com/playlist?list=PLE7TQZT0yYNgfcVHroweD19q5KR00SM>
[Dk](#)

3. Chrome web store:

<https://chrome.google.com/webstore/detail/vcat/gjbnjhimmkpplfccacmmkglojjgmkld>

References

- 1 <https://www.washington.edu/doi/book/export/html/1387> (Accessed :July/2019)
- 2 <https://www.schoolhealth.com/maltron-single-finger-mouth-stick-keyboard> (Accessed :July/2019)
- 3 <https://www.cio.com/article/2404851/the-magical-world-of-tablets-and-touchscreens.html> (Accessed :July/2019)
- 4 <https://www.cnn.com/2016/12/22/health/als-steve-saling-residence/index.html> (Accessed :July/2019)
- 5 <https://developer.amazon.com/en-US/alexa/alexa-skills-kit/asr> (Accessed :July/2019)
- 6 <https://equalentry.com/using-cortana-and-speech-recognition-together-on-windows-10/> (Accessed :July/2019)
- 7 <https://developer.apple.com/documentation/speech> (Accessed :July/2019)
- 8 <https://www.icse2018.org/track/icse-2018-Posters> (Accessed :July/2019)
- 9 <http://ranger.uta.edu/~csallner/papers/Ramachandra18Poster.pdf>
- 10 https://www.uta.edu/engineering/_downloads/Participant%20brochure%20web.pdf
- 11 <https://www.explainthatstuff.com/voicerecognition.html> (Accessed :July/2019)
- 12 <https://blog.nishtahir.com/2015/09/19/fuzzy-string-matching-using-cosine-similarity/> (Accessed :July/2019)

- 13 <https://www.w3.org/WAI/standards-guidelines/aria/> (Accessed :July/2019)
- 14 https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Techniques
(Accessed :July/2019)
- 15 https://www.w3.org/TR/wai-aria-1.1/#states_and_properties (Accessed :July/2019)
- 16 <https://cs.fit.edu/~kgallagher/Schtick/Serious/SWEBOKv3.pdf> (Accessed :July/2019)
- 17 <https://www.seleniumhq.org/>(Accessed :July/2019)
- 18 <https://www.soapui.org/> (Accessed :July/2019)
- 19 <https://www.learnqtp.com/descriptive-programming-simplified/> (Accessed :July/2019)
- 20 <https://www.guru99.com/introduction-to-selenium.html> (Accessed :July/2019)
- 21 <https://www.javatpoint.com/selenium-webdriver-architecture> (Accessed :July/2019)
- 22 <https://docs.seleniumhq.org/download/> (Accessed :July/2019)
- 23 <http://chromedriver.chromium.org/downloads> (Accessed :July/2019)
- 24 <https://www.edureka.co/blog/webelement-in-selenium/> (Accessed :July/2019)
- 25 <https://www.guru99.com/accessing-forms-in-webdriver.html> (Accessed :July/2019)
- 26 https://developer.chrome.com/extensions/api_index (Accessed :July/2019)
- 27 <https://www.json.org/> (Accessed :July/2019)
- 28 <https://www.w3.org/WAI/fundamentals/accessibility-intro/#what>(Accessed :July/2019)
- 29 <https://www.w3.org/TR/UNDERSTANDING-WCAG20/conformance.html> (Accessed
:July/2019)
- 30 <https://www.interactiveaccessibility.com/accessibility-statistics> (Accessed :July/2019)

31 <http://gs.statcounter.com/> (Accessed :July/2019)

32 <https://developers.google.com/web/updates/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API> (Accessed :July/2019)

33 <https://chrome.google.com/webstore/detail/lipsurf-voice-control-for/lnnmjmalakahagblkkcnjkoaihlfglon?hl=en>

34 <https://chrome.google.com/webstore/detail/click-by-voice/dleijbbjajmfcaiiadgjpgfjmfdfen?hl=en>

35 <https://chrome.google.com/webstore/detail/hands-free-for-chrome/ddgmnkioeodkdacpjbmlmihodjgmebnld?hl=en>

36 <https://chrome.google.com/webstore/detail/voice-actions-for-chrome/hhpjefokaphndbbidpehikcjhldaklje?hl=en>