COMPUTER ENGINEERING FOUNDATIONS, CURRENTS, AND TRAJECTORIES COLLECTION

Lisa MacLean, *Editor*

Advanced Selenium Web Accessibility Testing

Software Automation Testing Secrets Revealed

Narayanan Palani

MOMENTUM PRESS
ENGINEERING

ADVANCED SELENIUM WEB ACCESSIBILITY TESTING

ADVANCED SELENIUM WEB ACCESSIBILITY TESTING

Software Automation Testing Secrets Revealed

NARAYANAN PALANI

MOMENTUM PRESS
ENGINEERING

MOMENTUM PRESS, LLC, NEW YORK

Advanced Selenium Web Accessibility Testing: Software Automation Testing Secrets Revealed

First published by Momentum Press®, LLC
222 East 46th Street, New York, NY 10017
www.momentumpress.net

ISBN-13: 978-1-94944-943-3 (print)
ISBN-13: 978-1-94944-944-0 (e-book)

Momentum Press Computer Engineering Foundations, Currents, and Trajectories Collection

Cover and interior design by Exeter Premedia Services Private Ltd., Chennai, India

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Dedicated to

Lakshmi

My Grand Mother

For teaching me important lessons including the strong need of accessibility for everyone.

Abstract

If you are searching a topic on Google or buying a product online, web accessibility is a basic need! If a web page is better accessible through mouse and complex to navigate with keyboard are extremely difficult for users with disabilities. Web Accessibility Testing is a most important testing practice for customer facing web applications.

This book explains about the steps to write manual accessibility tests and convert them into automated selenium-based accessibility tests to run part of regression test packs. WCAG and Section 508 guidelines are considered across the book while explaining the test design steps.

Software testers with accessibility testing knowledge are most wanted for large organizations since the need to do manual and automated accessibility testing is growing rapidly! This book best illustrated the types of accessibility testing with test cases and code examples.

KEYWORDS

selenium testing; accessibility testing; web accessibility; pa11y; manual testing; automation testing; visual testing; WCAG; automated accessibility testing; non-functional testing

Contents

List of Figures

Foreword

"Web is for everyone."

As software engineers we are responsible to make a web inclusive for everyone, and everywhere by keeping accessibility design in mind. We are now in a world of agile and quick/point releases without compromising the quality of products. Ensuring an application is defect free in terms of accessibility is equally important as ensuring the same in terms of functionality. Testing application through a differently-abled person's perspective will be tough without proper knowledge on what is accessibility testing, and how it can be carried out using WCAG standards. Accessibility testing is one of the hot topics in the market and going to be the most wanted skill in future.

Narayanan has taken a brave step to explain and illustrate what is accessibility testing, and how it can be done 'right' using different manual as well as automation tools of choice.

Narayanan is a great author, and an excellent story teller which has been proven already through his publications and presentations. He has articulated accessibility and its needs along with the testing techniques so well in this book. This will definitely help the quality engineers and beginners who wish to step into the world of accessibility testing.

I know Narayanan for many years, and his contributions to testing community is highly remarkable. I would like to take this opportunity to thank Narayanan for all his effort to make the testing community better through his knowledge sharing and congratulations for the book release. Looking forward to seeing many such brilliant works from him!

Happy Learning!

Aparna Gopalakrishnan,
Technical Lead,
Wipro Technologies

Acknowledgment

Chandrashekar Korlahalli, Wipro Digital
Yamini Gupta, Wipro Digital

CHAPTER 1

Introduction: Accessibility Tester—A Great Futuristic Manual Testing Role

Web accessibility is an art of building the website for users with disabilities such as physical, vision, cognitive, and hearing disabilities. Remember that everyone of us will become disabled and need better accessibility in some way, sometime, in our lives naturally!

Approximately 1.3 billion people are affected by some form of visual impairment as per World Health Organization report, 'Blindness and vision impairment' dated 11 October 2018. Internet has changed people's life and most of the daily work has been heavily relying on websites for day-to-day activities such as accommodation arrangements, travel, food, and other basic necessities. Though websites are built with customer needs in mind, not all the websites are accessible for users with disabilities! Accessibility defects are very important and hard to replicate by testers who are not experienced such situations in their life.

Information Technology industry has been growing rapidly and there is a strong need for differently abled or physically challenged engineers to take a step forward and say what is wrong on the web pages they access in day-to-day life! When I met a friend, who has Group 2 Low vision on both eyes, he expressed his hard feelings on accessing a career website since he could not navigate to expected section of the web page using his mobile voice over. Similarly, another friend who is short could not understand ATM bank machine voice over to navigate to next screen when there is only limited text has been read from the voice recorder and It was difficult for him to see what is there displayed on ATM machine screen. This is extremely difficult for a normal tester to find these bugs whereas engineers with similar disabilities could easily catch these defects and log

them part of 'accessibility defects lists' hence there is a growing need to look for testers with disabilities in recent times. When I gave a demo of selenium based automated accessibility tests to group of engineers in Central London, United Kingdom, few among the audiences were actually attended the session to hire engineers for their software teams! Main challenge in finding right resource for accessibility testing is awareness of Web Content Accessibility Guidelines (WCAG) standards among software testers. Even though WCAG has been documented with great examples on websites, there is no much experienced project engineers on the ground to explain on the implementation. This book has been written in order to explain few accessibility test implementation possibilities on manual and automated testing. JavaScript has been used on automation test framework to showcase test examples throughout this book since NodeJS ecosystem is growing with various 'wonderful' test automation tools such as webdriverio (selenium), backstopJS and cucumberjs; Few examples are provided in java programming but the focus is more on 'accessibility test' rather than 'automated test.' If readers interested to learn more on automation test code examples, it is highly advisable to read my previous writings on 'Software Automation Testing Secrets Revealed part 1 and 2.'

1.1 FUTURE OF MANUAL TESTING

There are debates across the IT industry that the manual testing is going to go away and automation testing is going to take over most of the jobs! This is a true nature of future toward more test automation but manual testers knowing accessibility testing are expected to remain niche resources in near future!

Having compliant to international standards such as WCAG or section 508 is one of the mandatory needs for large customer facing organizations. When accessibility requirements are to be met, there is a strong need to hire manual testers who have great exposure toward accessibility and testing of such accessibility needs.

When automation testing tools such as selenium and UFT are in great demand, market also looking for manual testing who got exposures to JAWS or NVDA based accessibility testing.

Users with disability are more often using voice assistance such as JAWS or NVDA kind of tools to access the websites or applications on desktop in order to hear the voice from the respective websites. When using mobile phones, talkback is one of the famous options to hear the voice for the usage of particular website and access any applications from mobile.

There is a strong reason that not all the accessibility testing can be automated using tools such as selenium! Reason being, the experience of accessibility testing is really important when accessing the target application under test (AUT). This would explore more accessibility problems rather than relying on automated brainless tools to perform the actions for the test users.

1.2 HOW TO EXPERIENCE OR LEARN ACCESSIBILITY TESTING?

If you are interested to learn accessibility testing, it is as simple as searching 'NVDA' in Google and installing it on your desktop in order to start using it for every other keyboard movement on the website or AUT where accessibility needs to be verified!

Readers are recommended to watch a video related to NVDA with title 'NVDA: The free software empowering blind people world-wide' on YouTube URL: https://youtube.com/watch?v=Ks7AwV_uxO0

Once using NVDA or JAWS in desktop, kindly requesting readers to avoid using mouse or reduce the usage of mouse at least on the target AUT in order to verify the nature of the application.

1.3 ACCESSIBILITY TESTING

Users with disabilities such as visually impaired or 'physically challenged' (term used in this section as 'special users') are required accessible web applications for ease of use. This has been suggested as part of Web Content Accessibility Guidelines (known as 'WCAG')/General Services Administration Section 508 Standards (known as 'Section 508') in detail. Providing proper text for each section of the page and facilitating best user experience on the web pages are very important while providing accessibility features to web applications.

Note: Vision impairment has classification specified by Centers of Disease Control and Prevention (CDC) and it is not the only primary reason for performing accessibility tests on applications. Accessibility ensures the special users with disabilities can access the same web pages what normal user's access.

Some leading causes to visual impairment:

- Uncorrected refractive errors
- Cataract

- Diabetic retinopathy
- Corneal opacity
- Trachoma
- Macular degeneration (age based)
- Glaucoma

There is a common myth in accessibility testing is that the testing performed mainly for the customers having visual impairments. But it is not the only need to perform accessibility testing since following types of impairments are to be considered while testing:

- Auditory
- Speech
- Visual
- Cognitive
- Neurological
- Physical

Case study: Web access of www.dementiablog.org for patients with neurological disorders such as Alzheimer's disease, and so on.

Website has been validated using Total Validator Pro and holding down the CTRL and + or – will increase or decrease the font size of the page respectively and this information has been provided at: www.dementiablog.org/accessibility/

1.3.1 HOW TO KNOW WHEN ACCESSIBILITY TESTING IS REQUIRED FOR THE APPLICATION?

If the application is customer facing in any possible way, it is advisable to perform accessibility testing to get the application adhere to WCAG/ Section 508 standards.

1.3.2 HOW THE TESTING HAS BEEN DONE TO VERIFY THAT THE WEB APPLICATIONS ARE DEVELOPED WITH THOSE SPECIAL USERS IN MIND?

Web pages are tested by the way how differently abled customers access the pages using assistive technologies such as JAWS, NVDA, ZoomText, Windows Magnifier, Web Accessibility Toolkit.

Users with disabilities such as arthritis use keyboards to use Internet and they find it painful to use mouse. Similarly, wheelchair users having

difficulty in accessing computers, use switches to interact to web applications; If users having disability in touching the computer devices, they may even use nose or part of their fingers from hands or legs to access the web applications in technology devices such as computer, tablet, and so on. Some users with no vision, have no access or refrain from using a proper computer monitor since it is not required! In such situations, web applications should facilitate users with enough flexibility in using different features of rich media contents such as video play, pause, mute, and so on.

1.4 JAWS ASSISTIVE TECHNOLOGY

JAWS(licensed) is a famous tool used by visually challenged users across the world! Approximately 90 percent of web accessible special users are using JAWS to read out the web contents to interact with the applications. If a special user wants to login to a web page, JAWS is installed on the desktop and launched along with web login page using Internet Explorer (since JAWS works well with IE browser versions only). So, JAWS read out the login page contents when pressing keyboard TAB, DOWN or UP arrows and facilitating user navigation. If user want to open a web page in Firefox, it is preferable to use another tool known as NVDA (open source) which reads similar to JAWS for IE. If user wants a web page to open in chrome, NVDA works better but using Firefox for NVDA is most preferable.

1.5 JAWS INSTALLATION

Freedom Scientific (2008) JAWS Screen Reading Software [Web log post] Retrieved from https://freedomscientific.com/Downloads/JAWS

Note: Once JAWS has been installed and opened on the computer, it may prevent opening Google chrome or any other browser (except Internet Explorer); Similarly opened chrome browsers will get closed automatically since JAWS is opened for reading the screen. So, it is advisable to use Internet Explorer as preferred browser for web applications while using JAWS. If JAWS 18 throws error in installation with Windows 10 operating system, try installing older version such as JAWS 16.

1.6 TESTING LINKS ON THE WEB PAGES

If there is a link provided on particular web page to navigate to different page (in a new window), navigating to this link should read something similar to:

1.7 CLICKABLE LINK OPENS A NEW WINDOW

When JAWS read this content while navigating to the link will help special user to understand pressing ENTER using keyboard (or clicking on the link) leads to opening a new window and cursor or focus moves to new window!

Notes to developers:
HTML property: 'href' of a particular object facilitates JAWS/NVDA to read out the object as clickable. So, providing right text on 'title' or 'aria-label' fill facilitate a better accessibility to special users.

1.8 CONTENTS NEXT TO LINKS

If the focus is on the links, special users press DOWN key from keyboard to move to contents below those links on a given page. Since most of the special users rely on keyboard (not mouse) the page has to work as expected when navigating using keys from keyboard.

Notes to developers:
Title, aria-label or class of each object has been read out by JAWS/NVDA tools hence updating clear and relevant texts on aria-label for each object facilitate a better accessibility.

1.9 LISTED LINKS

JAWS users are familiar with keyboard shortcuts such as listed links functionality (INSERT+F7). These same keyboard shortcuts work in NVDA as well. Pressing on Insert+F7 on a web page opens a new window of JAWS and listing all possible links of the web page. So, user can select particular link and press ACTIVATE key from listed links which in turn perform click action on the link.

1.10 COLOR CONTRAST VERIFICATION

When visually impaired users access the web page, they prefer tools such as Zoomtext which zooms the page contents up-to 300 percent.[1]

[1] Partially impaired, less impaired users can access Zoomtext to view the page in short distance.

Verifying color contrast of objects within the web page is important and meeting 4.5:1 ratio of color contrast in normal mode and meeting 3:1 ratio in zoomed mode has been instructed by WCAG (for EU)/Section 508 (for USA)!

Note to QA:
If an image got color contrast of 4.3:1 in normal mode but it does match 3:1 ratio in 300 percent zoom, does that mean a defect or not?

Yes. It is a defect since it is not meeting minimum of 4.5:1 ratio in normal mode.

If an image got color contrast of 5:1 in normal mode but it does match 3:1 ratio in 250 percent zoom, does that mean a defect or not?

No

If an image appears on 'top left' of the page during web view but changes to 'top center' when zoomed to 300 percent in mobile view, does that mean a defect or not?

It depends on the specification really. If there is a specification to convert the page to mobile view after 250 percent zoom, it has to be instructed on the user story (of specification) on what to expect for the image and where it appears?

Reason: Mobile views are short and user can't see the web page in the same size of web view.

1.11 HEADERS IN ORDER

When special user opens a web page, URL has to be read and every TAB from user should leads to headers in order such as Header 1, Header 2, and so on.

Example: If the page reader (such as JAWS) reads of H3 after navigating from H1 which is incorrect to maintain HTML properties.

Case study: Mark has poor vision due to diabetic retinopathy and he is doing regular online shopping and added 15 products to the shopping cart using online portal of grocery retailer. When he decided to pay for the products, he wanted to cross check the list of items on the shopping cart and he navigated to shopping cart icon with the help of NVDA and keyboard actions, he managed to press ENTER which directed him to land in shopping cart page. When the page has been loaded, NVDA read the whole page contents in one go and he is trying to navigate to each product he wanted to read specifically. Unfortunately, the products are listed on the page as texts and he is unable to navigate to each product using TAB key or Listed Links option (using INSERT+F7).

Solution: Adding each product as a link on shopping cart or alternatively adding each product as header (2 or 3 based on the page hierarchy) would have made the life easy for Mark!

Testing Scope: If the shopping cart code has been altered by adding links or headers for products, that has to be tested across desktop, tablet and mobile devices to make sure the navigation is consistent and appropriate.

1.12 CONSTRUCTING MANUAL TEST CASES FOR ACCESSIBILITY TESTING

Test cases are suggested to focus on covering four principles of WCAG 2.0: Perceivable, Operable, Understandable, and Robust within test steps or test scenario level. Importantly, it has to cover success criteria such as A, AA, and AAA on Expected results.

Integrating accessibility testing part of agile testing projects.

When there is a user story developed and awaiting on sprint board* (in TO DO list), this has to be evaluated by business analyst/testers for accessibility testing scope.

Sprint Cycle discussed in this book:

BACKLOG > BLOCKED > TO DO > STORY KICK OFF > IN DESIGN > IN DEVELOPMENT > READY FOR TEST > IN TEST > DONE

If the story has been moved to 'Story kick-off' meetings, relevant WCAG/Section 508 guidelines has to be analyzed and discussed along with developers and stakeholders involved in the story kick off hence the accessibility test inputs goes to 'Acceptance Criteria' list!

Good Example:
If the user story is to develop a link as part of the agile project, the acceptance criteria can include following point:

-WCAG 2.4.4(Link Purpose) success criteria has to be met.

Bad Example:
Acceptance Criteria: 'Listed links has to be verified'
Reason:
When a link is developed, QA has to use JAWS to navigate to link and understand what has been read out within Internet Explorer; Similar test has to be conducted on NVDA using Firefox.

Similarly, QA use Listed Links to verify if the newly developed link appeared on the list and selecting it and pressing 'Activate' actually clicking on the link or not.

Finally hover over the link and see what text has been appeared on the hover over text (html property: 'title' of the object would have been displayed as 'hover over text').
In Mobile devices:
Enable voice settings to navigate to link and listen to the text has been read out.

Hence testing only listed links would not prove a complete test!

Early test automation on agile testing: Pa11y/Axe/Total Validator/Fire eyes can be used to scan the whole page when the full page of some part of the page has been developed! So, the respective 'user story' has to be tested (when it is moved from 'In Development' to 'Ready for Test') with Pa11y to scan the page and provide errors, warnings and recommendations.

Challenge: Pa11y helps in scanning particular page when URL has been provided. If the application is to be scanned after login or authentication, pa11y won't help as of today.

Solution1: it has to be scripted part of another tool called pa11y-ci hence the login part taken care of java program and then scanning particular page after authentication can be instructed on code level.

Solution2 (Preferable): AXE can navigate to particular page and scan the elements against WCAG /Section 508 standards after authentication.

1.12.1 EXAMPLE OF A TEST CASE IN GOOD FORMAT

Test Case Name: 01_NFT_Accessibility_FFXNVDA_Verify_Google_Search.

Test Description: H44 and H32 WCAG standards are verified on the web page www.google.com using Firefox and NVDA.

Pre-requisite: Install NVDA (open source) and Firefox quantum 58.0.2 on the Windows 10.

Test Step 1: Open NVDA on a Windows 10 operating system and use earphone to hear the voice provided by NVDA.

Expected Result 1: NVDA should be opened successfully and contents read out clearly by NVDA to user.

Test Step 2: Open Firefox Quantum (version 58.0.2 (64-bit)) on a Windows 10 operating system.

Expected Result 2: Firefox should be opened successfully.

Test Step 3: Launch www.google.com website.

Expected Result 3: Google search page should be opened and URL should be read successfully by NVDA.

Test Step 4: Navigate from URL to text box using TAB key of keyboard on the Google search page.
Expected Result 4: NVDA should read the details of the text box.
Test Step 5: Enter text within the text box.
Expected Result 5: Text entered on the text box should be read out clearly.
Test Step 6: Press TAB key to navigate to Search Button.
Expected Result 6: Submit button should be read clearly.
Test Step 7: Press ENTER on the Search button.
Expected Result: NVDA should read the search results clearly.

1.12.2 BAD TEST CASE

Test Case Name: 01_NFT_Accessibility_IEXNVDA_Verify_Google_Search.

Test Description: H44 and H32 WCAG standards are verified on the web page www.google.com using Internet Explorer 11 and NVDA.

Reason:
NVDA doesn't go well with Internet Explorer hence it is advisable to use JAWS for this test case to verify accessibility on Internet Explorer.

CHAPTER 2

SHIFT LEFT TESTING

Performing software testing during agile sprint-based developments and moving testing activities to early stages of development is a best practice and trend across testing industry from the year 2017.

2.1 EARLY SHIFT LEFT QUALITY ASSURANCE

Stages of Agile sprint-based development are split across six stages to describe examples in this book. Every sprint varies from two weeks to four weeks based on team decisions but majority of the sprint-based teams follow similar phases of development in their teams hence following stages are provided to explain how best accessibility testing can be tested early in life cycle?

TO DO: Analyze the level of accessibility testing needs for the story when it listed under 'TODO' list.

STORY KICK OFF: Discuss the types of accessibility testing needed and update WCAG/Section 508 classifications (to be tested) on acceptance criteria of the user story (by BA/testers).

IN DESIGN: Design in line with WCAG/Section 508 standards (developers/design engineers); Design accessibility test combinations (testers).

IN DEVELOPMENT: Develop the html properties to facilitate better accessibility and perform local tests in development boxes using JAWS or NVDA (by developers); Develop manual accessibility test cases(testers).

READY FOR TEST: QAs Scan the page using Pa11y (if it is single page accessible by URL) or AXE (if the page is located after authentication or after multiple page navigations/actions). Once scanned provide the scan report and discuss along with developers to remove

false positives identified during page scan. Once issues resolved at this stage, the application under test (AUT) is actually ready for test execution hence move forward to 'IN TEST.'

IN TEST: Execute the manual accessibility test cases using JAWS, NVDA, Zoomtext and other tools to validate the accessibility of the AUT. Once manual test cases executed and test failures(defects) are fixed, automate the subset (up to 30 percent) of tests using webdriverio for 'Automated Accessibility Tests' to run as part of regression tests. So, QA need not have to rerun the manual test case on the particular story when automated tests run and identify defects as part of CI/CD pipeline! Once manual test execution completed/automated tests are created and executed at least once/defects are fixed, prove the test evidence by discussing with Business Analyst, Product Owners and Developers Hence Scrum Master or Business Analyst or Product Owner can move the user story to DONE (based on Definition of Done (DoD) and Responsibilities within the team).

DONE: Story moves to done and the regression accessibility automated tests run from here on!

Automated accessibility tests.

Sample Test written in Selenium webdriverIO using JavaScript:

```
var qa-selector = 'q';
var attribute ='default-theme des-mat';

driver.findElement(selenium.By.id(qa-selector))
.then(function (element) {
element.sendKeys(Key.SPACE);
return element;
})
.then (function(element) {
Return element.getAttribute('class')
})
.then (function(attribute){
Expect (attribute).toEqual('true');
});
```

Wide range of YouTube videos available on accessibility testing:

Narayanan, P. March 15, 2018. "Accessibility Testing Playlist." *Web log post*. *Retrieved from* https://youtube.com/playlist?list=PLXW5SOJAYvfeNliJ-Dy6P74RaPuzXlyx_A

Alternative Short URL: https://goo.gl/bQdVRX

2.2 ACCESSIBILITY TESTING IN AGILE PROJECTS

When projects are developed completely in agile methodology, it is advisable to start implementing accessibility tests early in life cycle. Following are some of the examples that proves the step by step move of accessibility tests from right most end of delivery to left most end of story discussion!

Step 1:
Test the target AUT during 'In Test Phase':

Backlog	To Do	Story Kick-off	In Design	In Development	In Test	Done
					Manual Accessibility Testing Accessibility Scanning	

Advantages:

- AUT can be developed fast and deployed for testing.
- AUT is available for accessibility testing and defects can be raised to track those accessibility issues.
- AUT can be redeployed after fixing majority of accessibility testing issues identified part of testing process to retest and perform regression tests.

Disadvantages:

- AUT is not scanned by tools such as axe or Pa11y earlier in life cycle and this cause list of defects which are getting introduced during 'In Test.' This should have been prevented if the application scanned for accessibility defects during 'In Development' phase.
- Performing retesting of accessibility during 'In Test' after defect fixes are highly manual, time consuming and tedious process.
- If there are design issues (such as header 1 missing or sequence of information is incorrect due to complex design, and so on) can be retained part of code since the design phase has been crossed already.
- Story living part of 'In Test' can be longer than usual due to list of defects raised related to accessibility.

Step 2:
Scan AUT during 'In Development Phase' and test the target AUT during 'In Test Phase':

Backlog	To Do	Story Kick-off	In Design	In Development	In Test	Done
				Accessibility Scanning	Manual Accessibility Testing	

Advantages:

- AUT scanned by tools such as axe or Pa11y earlier in life cycle.
- AUT can be developed fast and scanned to identify and fix any accessibility related issues such as color contrast errors and deployed for testing by preventing those accessibility testing issues in 'Test' phase hence less defects are expected.
- AUT can be redeployed after fixing majority of accessibility testing issues which are identified part of testing process (post application scan in development phase) to retest and perform regression tests.

Disadvantages:

- Performing retesting of accessibility during 'In Test' after defect fixes are highly manual, time consuming and tedious process.
- If there are design issues (such as header 1 missing or sequence of information is incorrect due to complex design, and so on) can be retained part of code since the design phase has been crossed already.
- Story living part of 'In Test' can be longer than usual due to list of defects raised related to accessibility.

Step 3:
Scan AUT during 'In Development Phase' and test the target AUT within development cycle with tools such as JAWS or NVDA, WAT and other tools by cherry picking possible sub set of accessibility tests and test remaining tests during 'In Test Phase':

Backlog	To Do	Story Kick-off	In Design	In Development	In Test	Done
				Accessibility Scanning Subset of manual Accessibility Testing	Remaining set of manual Accessibility Testing	

Advantages:

- AUT scanned by tools such as axe or Pa11y earlier in life cycle.
- AUT can be developed fast and scanned to identify and fix any accessibility related issues such as color contrast errors, tested with the help of first subset of tests to identify early defects to fix quick and getting deployed for testing by preventing those accessibility testing issues in 'Test' phase hence extremely less defects are expected.
- AUT can be redeployed after fixing majority of accessibility testing issues which are identified part of testing process (post application scan and test in development phase) to retest and perform regression tests.

Disadvantages:

- Performing retesting of accessibility during 'In Test' after defect fixes are highly manual, time consuming, and tedious process.
- If there are design issues (such as header 1 missing or sequence of information is incorrect due to complex design, and so on) can be retained part of code since the design phase has been crossed already.
- Story living part of 'In Test' can be longer than usual due to list of defects raised related to accessibility.

Step 4:
Discuss during 'In Design' about design changes required to match accessibility standards, scan AUT during 'In Development Phase' and test the target AUT within development cycle with tools such as JAWS or NVDA, WAT and other tools by cherry picking possible sub set of accessibility tests and test remaining tests during 'In Test Phase':

Backlog	To Do	Story Kick-off	In Design	In Development	In Test	Done
			Matching design with Accessibility Standards	Accessibility Scanning Subset of manual Accessibility Testing	Remaining set of manual Accessibility Testing	

Advantages:

- AUT can be designed by keeping alignment of headers or sequence of information in mind.

- AUT scanned by tools such as axe or Pa11y earlier in life cycle.
- AUT can be developed fast and scanned to identify and fix any accessibility related issues such as color contrast errors, tested with the help of first subset of tests to identify early defects to fix quick and getting deployed for testing by preventing those accessibility testing issues in 'Test' phase hence extremely less defects are expected.
- AUT can be redeployed after fixing majority of accessibility testing issues which are identified part of testing process (post application scan and test in development phase) to retest and perform regression tests.

Disadvantages:

- Performing retesting of accessibility during 'In Test' after defect fixes are highly manual, time consuming, and tedious process.
- Story living part of 'In Test' can be longer than usual due to list of defects raised related to accessibility.

Step 5:
Update Acceptance Criteria of stories during kick-off and discuss with developers and testers on what are the relevant accessibility standard numbers can be considered while progressing on the story and get the actual effort estimation for story points which includes the efforts required for accessibility testing.

Backlog	To Do	Story Kick-off	In Design	In Development	In Test	Done
		Update 'Acceptance Criteria' to match relevant WCAG or Section 508 standards and include Accessibility in mind for 'Story Points' calculation	Matching design with Accessibility Standards	Accessibility Scanning Subset of manual Accessibility Testing	Remaining set of manual Accessibility Testing	

Advantages:

- AUT can be designed by keeping alignment of headers or sequence of information in mind.

- AUT scanned by tools such as axe or Pa11y earlier in life cycle.
- AUT can be developed fast and scanned to identify and fix any accessibility related issues such as color contrast errors, tested with the help of first subset of tests to identify early defects to fix quick and getting deployed for testing by preventing those accessibility testing issues in 'Test' phase hence extremely less defects are expected.
- AUT can be redeployed after fixing majority of accessibility testing issues which are identified part of testing process (post application scan and test in development phase) to retest and perform regression tests.

Disadvantages:

- Performing retesting of accessibility during 'In Test' after defect fixes are highly manual, time consuming and tedious process.
- Story living part of 'In Test' can be longer than usual due to list of defects raised related to accessibility.

Step 6:
Implement automated accessibility testing:

Backlog	To Do	Story Kick-off	In Design	In Development	In Test	Done
		Update 'Acceptance Criteria' to match relevant WCAG or Section 508 standards and include Accessibility in mind for 'Story Points' calculation	Matching design with Accessibility Standards	Accessibility Scanning Subset of manual Accessibility Testing	Remaining set of manual Accessibility Testing, Implement 'automated' accessibility testing scripts	

Advantages:

- AUT can be designed by keeping alignment of headers or sequence of information in mind.
- AUT scanned by tools such as axe or Pa11y earlier in life cycle.
- AUT can be developed fast and scanned to identify and fix any accessibility related issues such as color contrast errors, tested with the

help of first subset of tests to identify early defects to fix quick and getting deployed for testing by preventing those accessibility testing issues in 'Test' phase hence extremely less defects are expected.
- AUT can be redeployed after fixing majority of accessibility testing issues which are identified part of testing process (post application scan and test in development phase) to retest and perform regression tests.
- Automated accessibility tests will verify any accessibility issues part of Continuous Integration and Delivery pipeline.
- No need of repeating any manual accessibility tests after performing initial round of manual accessibility verification and validation.

Disadvantages:

- N/A

Step 7:
Implement automated accessibility testing in life cycle as earlier as possible:

Backlog	To Do	Story Kick-off	In Design	In Development	In Test	Done
		Update 'Acceptance Criteria' to match relevant WCAG or Section 508 standards and include Accessibility in mind for 'Story Points' calculation	Matching design with Accessibility Standards	Accessibility Scanning Subset of manual Accessibility Testing Start writing 'automated' accessibility testing scripts	Remaining set of manual Accessibility Testing, Finish writing 'automated' accessibility testing scripts	

Advantages:

- AUT can be designed by keeping alignment of headers or sequence of information in mind.
- AUT scanned by tools such as axe or Pa11y earlier in life cycle.
- AUT can be developed fast and scanned to identify and fix any accessibility related issues such as color contrast errors, tested with the help of first subset of tests to identify early defects to fix quick and getting deployed for testing by preventing those accessibility testing issues in 'Test' phase hence extremely less defects are expected.

- AUT can be redeployed after fixing majority of accessibility testing issues which are identified part of testing process (post application scan and test in development phase) to retest and perform regression tests.
- Automated accessibility tests will verify any accessibility issues part of Continuous Integration and Delivery pipeline.
- No need of repeating any manual accessibility tests after performing initial round of manual accessibility verification and validation.

Disadvantages:

- N/A

Some FAQs:

2.2.1 CAN WE COMPLETELY SKIP MANUAL TESTING PART OF ACCESSIBILITY?

It is a wrong approach since experiencing the way how the real application has been used by differently abled users are very important hence manual testing needs to be done at least once.

2.2.2 IS IT TESTER'S RESPONSIBILITY TO SCAN THE AUT USING AXE OR PA11Y?

It is tester's role to suggest what needs to be scanned and help developers to understand what are some of the valid issues from the complete list of violation thrown by tools such as axe, pa11y, or other toolsets.

2.2.3 CAN WE REQUEST DEVELOPERS TO DO THIS MANUAL TESTING?

It is not advisable.

2.2.4 CAN WE SUGGEST TESTERS TO WRITE AUTOMATED ACCESSIBILITY TESTS BEFORE 'IN TEST' PHASE?

It is possible only when the tester has been experienced testing the application for accessibility standards and automated accessibility verification steps part of earlier sprints. But it is possible to automate only when first

round of manual accessibility testing has been completed successfully with complete defect fixes and retests. Reason being, manual tests open up the idea to provide broader coverage to those automated accessibility tests.

2.2.5 IF PAGE IS SCANNED USING AXE, IS IT CONSIDERED THAT ACCESSIBILITY TESTING COMPLETE?

No-it is just one of the ways to initiate accessibility validations but accessibility testing is not started until performed a full round of manual accessibility testing followed by automated test script development of some tests (which are tested manually) hence these tests can run in CI/CD pipeline regularly to prove that the accessibility is validated as part of every code change.

CHAPTER 3

TEST COVERAGE

Covering(writing) all the test requirements through test cases to make sure that test coverage is 100 percent (also known as 'Requirement Traceability') is a best practice to verify and validate every possible testable item in Application Under Test (known as 'AUT').

When testers planning for accessibility testing, WCAG and Section 508 are the primary guidelines to match the need or the base for the test cases coverage by looking at each requirement. Similar to 'Requirement Traceability Metrics,' there can be a 'Guidelines Traceability Metrics' to match multiple test cases to each requirement and WCAG guidelines to prove complete test coverage.

Example:

Each requirement can be mapped to new test scenarios by covering one or multiple guidelines in the following table. Alternatively, tests can be written according to match the device coverage and assistive technologies such as JAWS or NVDA hence they can provide a good coverage across WCAG guidelines and test requirements.

Guidelines	**Test Coverage**									
		Devices					**Assistive Technology**			
WCAG 2.0 Success Criteria	**Section 508**	**Computer-Windows**	**Computer-Safari**	**Mobile-Android**	**Mobile-iOS**	**Tablet**	**JAWS**	**NVDA**	**Others**
1.1.1 Non-text Content [A]	1194.22(a)								
1.2.1 Pre-recorded Audio-only and Video-only [A]	1194.22(a)								

(*Continued*)

1.2.2 Captions (Pre-recorded) [A]	1194.22(b) and .24(c)
1.2.3 Audio Description or Media Alternative (Pre-recorded) [A]	1194.22(b) and .24(d)
1.2.4 Captions (Live) [AA]	1194.22(b) and .24(c)
1.2.5 Audio Description (Pre-recorded) [AA]	1194.22(b) and .24(d)
1.3.1 Information and Relationships [A]	1194.22(e) through (h)
1.3.2 Meaningful Sequence [A]	None
1.3.3 Sensory Characteristics [A]	None
1.4.1 Use of Color [A]	1194.21(i) and .22(c)
1.4.2 Audio Control [A]	None
1.4.3 Contrast (Minimum) [AA]	None
1.4.4 Resize Text [AA]	None
1.4.5 Images of Text [AA]	1194.21(f)
2.1.1 Keyboard [A]	1194.21(a)
2.1.2 No Keyboard Trap [A]	None
2.2.1 Timing Adjustable [A]	1194.22(p)
2.2.2 Pause, Stop, Hide [A]	1194.21(h)
2.3.1 Three Flashes or Below Threshold [A]	1194.21(k) and .22(j)
2.4.1 Bypass Blocks [A]	1194.22(o)
2.4.2 Page Title [A]	1194.22(i)
2.4.3 Focus Order [A]	None
2.4.4 Link Purpose (In Context) [A]	None
2.4.5 Multiple Ways [AA]	None
2.4.6 Headings and Labels [AA]	None

2.4.7 Focus Visible [AA]	1194.21(c)
3.1.1 Language of Page [A]	None
3.1.2 Language of Parts [AA]	None
3.2.1 On Focus [A]	1194.21(l) and .22(n)
3.2.2 On Input [A]	1194.21(l) and .22(n)
3.2.3 Consistent Navigation [AA]	None
3.2.4 Consistent Identification [AA]	1194.21(e)
3.3.1 Error Identification [A]	1194.21(l) and .22(n)
3.3.2 Labels or Instructions [A]	1194.21(l) and .22(n)
3.3.3 Error Suggestion [AA]	None
3.3.4 Error Prevention (Legal, Financial, Data) [AA]	None
4.1.1 Parsing [A]	None
4.1.2 Name, Role, Value [A]	1194.21(d)

3.1 TEST COVERAGE GUIDELINES AND TEMPLATES

Requirement traceability matrix are designed to map test scripts/cases to link to respective test requirements in order to understand any coverage gaps hence requirement traceability has been ensured.

Similarly, requirements and accessibility standards are mapped/linked to test cases to understand if any relevant accessibility standards and requirements are missed!

Example:

Requirement ID	WCAG Accessibility Standard	Accessibility Tests
XXX	*Non-text Content: Understanding SC 1.1.1*	1.1.1_1.3_NFT_Accessibility_FFXNVDA_Verify_Github_website
XXX	*Adaptable: Understanding Guideline 1.3*	1.1.1_1.3_NFT_Accessibility_FFXNVDA_Verify_Github_website
XXX	*Distinguishable: Understanding Guideline 1.4*	1.4_NFT_Accessibility_IEXWAT_Verify_Github_website

Complete guidelines of WCAG can be found here: https://w3.org/WAI/standards-guidelines/wcag/

3.2 ACCESSIBILITY TEST COVERAGE CHECKLIST-STORY KICK-OFF STAGE

Questions/Checklist for the User Stories during 'Kick Off' or 'To-Do' stage	Mandatory (M)/ Optional (O)
Has Acceptance Criteria (AC) section updated with 'Accessibility Guideline' number to be fulfilled? Ex: WCAG 1.1.1 has to be met	M
Each acceptance criteria of accessibility have been assigned to respective developer or tester?	O
AC: Accessibility code scan (using AXE or PA11Y) to be performed by developer?	O
AC: Accessibility Test Cases should be designed and completed before story moved to 'Test'	M
AC: Automated accessibility tests are to be designed once manual testing is completed	O
AC: Outcome of test execution as Recommendations/Issues/Defects	O
Definition of Done: Definition of Accessibility Defect Severity 1,2,3	M

Definition of Done: Defining when to complete accessibility test execution	M
Definition of Done: Procedure to 'Defer' Accessibility Defects to fix it or consider it in the future fixes and steps to raise 'Waivers'	M

3.3 ACCESSIBILITY TEST COVERAGE CHECKLIST-DESIGN STAGE

Questions/Checklist for the User Stories in Design Stage	**Mandatory (M)/ Optional (O)**
There should not be any 'keyboard traps' on any of the sections part of user story; In order to avoid keyboard traps, designers are suggested to provide navigation flow of forms to developers	M
Application should not fully rely on animation. Animation content should be available with text alternatives	M
Application should be built with logical start and end states?	M
If application use motion, avoid using motion and try using dissolve transitions	M
Color selections should be associated with text alternatives	M
If application contains flash or animation or video kind of audio/video components, used simulation spectacles to test the application? Example: Vision Simulation Spectacles for: • Cataracts and Low Acuity • Tunnel Vision • Tunnel Vision and Low Acuity • Hemianopia • Diabetic Eye Disease • Macular Degeneration **Reference** http://optimalowvision.co.uk/product.cfm?prod=474&dept=479	O

(Continued)

If focus indicators and links are available on web application, has the application been tested using hand magnifiers?	O
Are these questions answered by designers? • What is the overall purpose of this visual? • What is the best way to convey that? • Which parts of visual is decorative and need text alternative? • Which parts of images are functional images such as left, right arrows which needs text alternative?	M
Audio descriptions are provided to media part of applications? Will that help deaf audiences to read and understand about the video?	M
Every form fields such as check boxes and radio boxes are provided with a <label>?	M
If application contains placeholders (such as Date of Birth field with grey colored texts as DDMMYYYY which disappear when the field is focused), has this placeholder designed to display below, above or right-hand side of the field rather than hiding it when focused?	M

3.4 KEYBOARD SHORTCUT VERSUS TEST CASE COVERAGE MATRIX

Test cases are to be mapped against each keyboard shortcuts supported by assistive technologies such as JAWS and NVDA.

JAWS keyboard shortcuts can be found at: https://webaim.org/resources/shortcuts/jaws

NVDA keyboard shortcuts can be found at: https://webaim.org/resources/shortcuts/nvda

3.5 SEVEN ACCESSIBILITY REQUIREMENTS CONSIDERATIONS FOR BETTER TEST COVERAGE

While testing web applications or documents it is highly advisable to consider following areas of testing and generate test cases:

1. Structure
2. Figures
3. Hyperlinks
4. Lists
5. Columns
6. Color and Contrast
7. Tables

If testers can map their test cases to all these seven areas of web application or document, it would feasible to catch handful of defects in test case steps. The next step is to update test cases to cover maximum possibilities to validate these seven areas within the AUT.

REFERENCE

U.S. Access Board. 2017. "Comparison Table of WCAG 2.0 to Existing 508 Standards." *Web Log Post*. Retrieved from https://access-board.gov/guidelines-and-standards/communications-and-it/about-the-ict-refresh/background/comparison-table-of-wcag2-to-existing-508-standards

CHAPTER 4

TOOLS INSTALLATION

4.1 AXE CORE ACCESSIBILITY ENGINE

Installation
GitHub 2018. "Accessibility Engine for Automated Web UI Testing." *Web log post*. Retrieved from https://github.com/dequelabs/axe-core
Update package.json file:

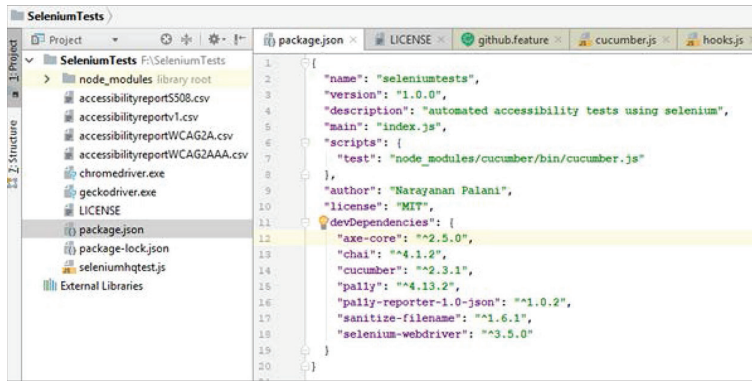

Figure 4.1. Selenium package.json file opened from WebstormIDE.

Install from terminal:
npm install axe-core --save-dev

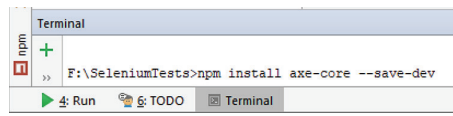

Figure 4.2. Terminal of Webstorm IDE tool.

Installation completion

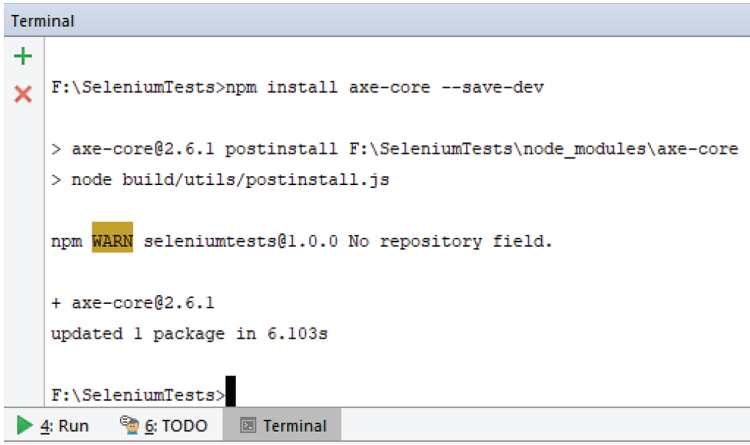

Figure 4.3. Terminal of WebstormIDE after installation of axe core through npm installation.

4.2 AXE PLUGIN FOR GOOGLE CHROME BROWSER

Search in www.google.com for axe plugin which can be used along with Google Chrome:

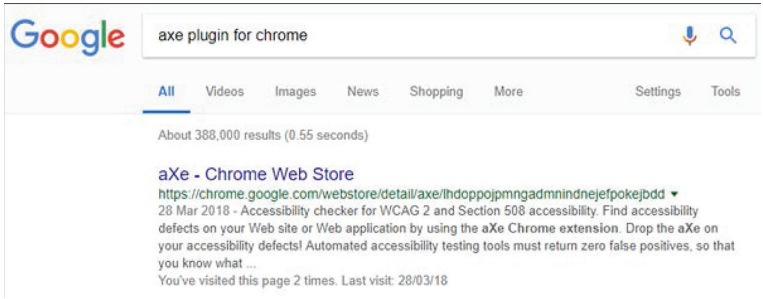

Figure 4.4. Google search on Axe plugin for Chrome.

Click on 'Add to Chrome' to add this plugin to Chrome browser:

Figure 4.5. Axe Plugin for Google Chrome browser.

When a page needs to be scanned, please click on F12 (or Fn+F12) to open developer tools to navigate to axe tab:

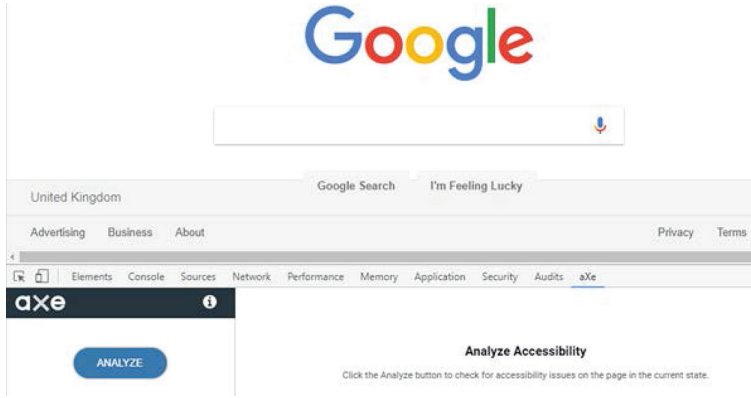

Figure 4.6. Press F12 from Google Chrome to see axe as a tab and click on analyze to scan the page to find accessibility violations on this page.

Click Analyse button on the axe tab:

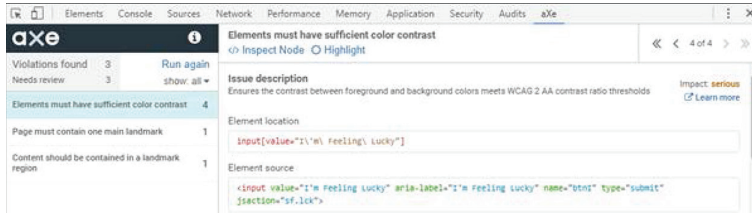

Figure 4.7. Axe Google Chrome plugin provides the violation list of web page.

Analyze the issues one by one to avoid false positives (issues that are not required any investigation or incorrect notification by axe).

4.3 ACHECKER TOOL ONLINE

Visit achecker.ca website and please enter the website to check any accessibility issues. For example, please enter https://github.com and press enter (under web page URL).

This will provide list of known issues from the website based on WCAG standards in few seconds. This tool will help to check accessibility issues on a single page;

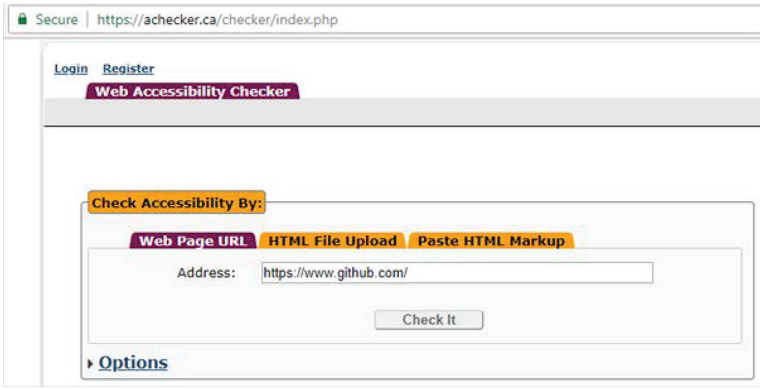

Figure 4.8. Achecker website to scan the web page on accessibility validations.

Recommendation:
If the same accessibility issues to be checked after login page or a page which gets visible after a scenario, it is advisable to try scanning 'axe' plugin.

CHAPTER 5

WEB ACCESSIBILITY TESTING

Special users with disabilities trying to access a particular web page to book tickets or submit insurance claims or any regular online activities needed full support of keyboards while using web pages and assuming that majority of those users do not use mouse to access the web applications. When the web pages don't support or partially work with keyboard leave such special users in challenging condition. There are keyboards designed specific to their needs and available in the market; When users access screen readers and keyboards, there are in a position to listen to the information what available on the web application and perform navigations, actions and events using their keyboards. For example, users reading a web page of google.com to perform search on particular information regularly using a device such as desktop or laptop with access to keyboard and mouse; Users with disabilities use screen reading software's such as JAWS/NVDA and launch the browsers using keyboard keystrokes and type the website what intended and it is read out by such software to users hence users with disabilities basically hear the information from the page and press keyboard for keystrokes such as Tab, Shift + Tab (lead to navigate backwards), Left, Right, Up or Down arrows, and so on.

5.1 TESTING APPROACH

During functional/regression testing, web applications are tested against user stories in agile projects where as projects in waterfall model or V model are facilitated with requirement specifications or formal requirement documents to test against.

During accessibility testing, requirements are evaluated against accessibility standards such as WCAG or Section 508 standards and tested by keeping both (requirements and accessibility standards) as base references.

5.2 BROWSER COMPATIBILITY

During functional tests, widely used browsers of customer scope are identified and targeted as a scope for compatibility testing such as Internet Explorer, Safari, Firefox, Chrome, and so on. During accessibility testing, those customer preferred browsers are validated using respective 'compatible' screen reading software's which are used by the users with disabilities. Until 2017, JAWS has been widely used by users with disabilities and it is highly compatible to Internet Explorer browser. Similarly, some of the users with disabilities prefer to use NVDA as a screen reading software and it is highly compatible to Firefox browser. When web browsers are used from hand held devices such as mobile or tablets, respective android or iOS applications are facilitated with voice over facilities hence it has to tested using physical devices in order to understand how the web pages has been read by those voice reading applications when used from mobile or tablets.

CHAPTER 6

TESTING WCAG ACCESSIBILITY STANDARD—PERCEIVABLE

Web pages provide high capability to users to understand the information in every section leads to attracting more customers and better comfort when person with disability accessing the pages.

6.1 NON-TEXT CONTENT TESTING

When there is an image on the web page or any graphics other than texts are to be presented with text alternative since visually impaired users may not see the non-text contents and they will be in a position to understand it only when there is any text alternative.

6.2 SAMPLE TEST ON GITHUB.COM FOR NON-TEXT CONTENT VERIFICATION

Test Case Name: 1.1.1_1.3_NFT_Accessibility_FFXNVDA_Verify_Github_website
Test Description: Following WCAG standards are verified on the web page www.github.com using Firefox and NVDA

1.1.1 Non-text content

1.3 Adaptable (1.3.1 Info and Relationships/1.3.2 Meaningful Sequence/ 1.3.3 Sensory Characteristics)

Reference:

W3C. 2016. "Non-text Content: Understanding SC 1.1.1." *Web Log Post.* Retrieved from https://w3.org/TR/UNDERSTANDING-WCAG20/text-equiv-all.html

W3C. 2016. "Adaptable: Understanding Guideline 1.3." *Web Log Post*. Retrieved from https://w3.org/TR/UNDERSTANDING-WCAG20/content-structure-separation.html

Pre-requisite: Install NVDA (open source) and Firefox quantum 58.0.2 on the Windows 10.

Test Step 1: Open NVDA on a Windows 10 operating system and use earphone to hear the voice provided by NVDA

Expected Result 1: NVDA should be opened successfully and contents read out clearly by NVDA to user.

Test Step 2: Open Firefox Quantum (version 58.0.2 (64-bit)) on a Windows 10 operating system

Expected Result 2: Firefox should be opened successfully.

Test Step 3: Launch www.github.com website

Expected Result 3: GitHub page should be opened and URL should be read successfully by NVDA.

Test Step 4: Navigate from URL to text box using TAB key of keyboard on the Google search page

Expected Result 4: NVDA should read the details of the links, text boxes and any images should be read with the text alternatives.

Test Step 5: Enter text within the search text box

Expected Result 5: Text entered on the text box should be read out clearly.

Test Step 6: Press TAB key to navigate to the contents in proper sequence

Expected Result 6: Every object has to be read clear.

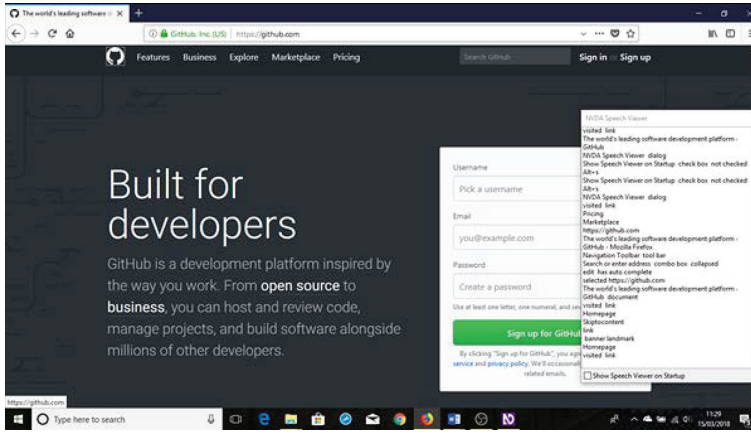

Figure 6.1. Accessing github website and using NVDA to read the contents on this website.

Test Step 7: Press INSERT + F7 key to navigate to the listed links

Expected Result 7: Listed Links should be displayed with complete links available on the page.

Actual Result:

GitHub logo has been read as 'Home Page visited link' hence there is a text alternative for the image provided as shown in the previous figure.

Pressing Insert+F7 opens up Elements list box with complete list of links available on the webpage:

Press Enter by selecting Home page which navigate to particular link and activates it hence the page has been validated for the following accessibility points:

- Non-text contents are provided with text alternatives
- Sequence of the contents are verified
- Keyboard keys such as Enter, Tab, Shift Tab, Left, Right, Up and Down arrows are verified for the navigation

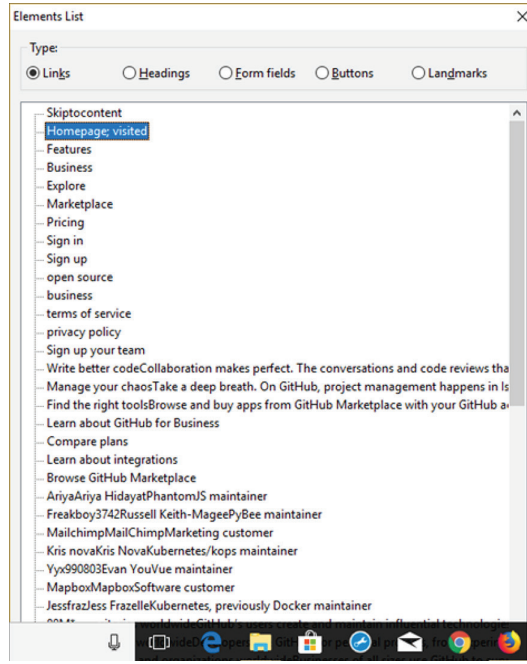

Figure 6.2. Elements list of the web page has been displayed on NVDA.

CHAPTER 7

Writing Automation Tests for Keyboard Actions Using Selenium

WebdriverIO (JavaScript version of selenium) provides a better feature to verify the keyboard actions and navigations within the web pages hence each navigation using Tab, Shift Tab can be checked on any given page during regular Behavior Driven Development (BDD) tests.

Example script on performing Tab key using Selenium WebdriverIO (JavaScript):

Feature: Searching for GitHub projects
As an internet user
In order to find out more about GitHub projects
I want to be able to search for information about webdriverio
Scenario: 1.3.2 WCAG sequence test-Tab from username to email
When I click on username
Then I press tab key to navigate to verify placeholder of email as you@example.com

Gherkin is the language of Behavior Driven Development (known as BDD) to write scripts with prefixes such as Given, When and Then! If tester wants to validate a simple login of a web page, it can be written in the following format:

Given I open github.com on Internet Explorer
When I enter valid **user name** and **password**
And I click **Login** button
Then I navigate to **home page**

In the previous example, 'And' in the third line also act like 'When'

Usually automated scripts come with respective 'step definition' for each line mentioned in the BDD and 'locators' identified and updated to the scripts to run on the page and find those objects to perform actions or events.

In this search feature example of GitHub website, user has to perform tab navigation on GitHub page to check if the navigation sequence is working as expected. So, the sample scenario has been provided to navigate from user name text box to email text box using tab key.

Step Definition:

```
When(/^I click on username$/, function (next) {
this.driver.get('https://github.com/');
this.driver.findElement(By.id('user[login]' ))
.click()
.then(function() {
next();
});
});

Then(/^I press tab key to navigate to verify (.*) of email as (.*)/,
function(attribute,expRes,next){

varelement_to='user[email]';
varelement_from='user[login]';
this.driver.findElement(By.id(element_from))
.sendKeys(Key.TAB);
this.driver.findElement(By.id(element_to)).getAttribute(attribute)
.then(function (readValue) {
expect(readValue).to.equal(expRes);
next();
});
});
```

Code example to write the same keyboard elements of Tab or Enter key press within Java programming-based Selenium Tests:

```
packageseleniumJavaFramework.KeyboardNavigationTest;
importjava.util.concurrent.TimeUnit;
importorg.openqa.selenium.By;
importorg.testng.annotations.BeforeTest;
importorg.testng.annotations.Test;
importorg.testng.annotations.AfterTest;
importorg.openqa.selenium.WebDriver;
importorg.openqa.selenium.Keys;
importorg.openqa.selenium.firefox.FirefoxDriver;

public class pressTabTest{
WebDriver driver = new FirefoxDriver();
@BeforeTest
public void setUp(){
driver.get("https://github.com/");
}
@Test
public void findElementbyXpath(){
WebElementqeElement = driver.findElement(By.xpath(".//*[@
id='user[login]']"))
qeElement.click();
qeElement.sendKeys(Keys.TAB);
qeElement.sendKeys(Keys.ENTER);
}
@AfterTest
public void doThis(){
driver.quit();
}
}
```

Note: It is possible to concatenate two key events such as Tab and Enter into single line of code as follows:

```
qeElement.sendKeys(Keys.TAB + Keys.ENTER);
```

Another example for Tab key in JavaScript program-based seleniumWebdriverIO:

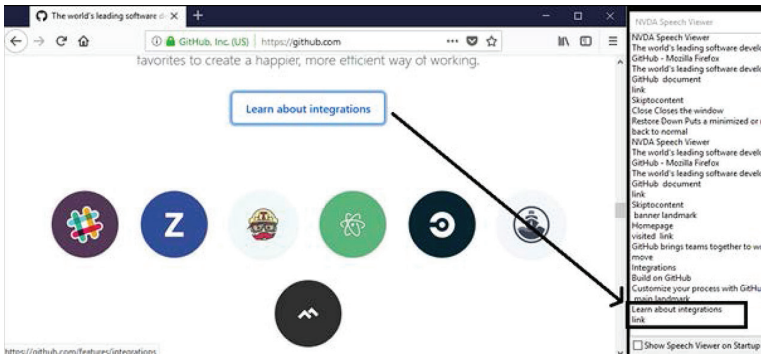

Figure 7.1. NVDA Speech Viewer reads about the link.

HTML of the object:
Open the website on Google Chrome browser, press F12 to open developer tools and then select 'Elements' tab to see the html properties of the page; As an alternative, right click on the object ('Learn about integrations' in this example) and select 'Inspect Element' to see:

```
<a href=”/features/integrations” class=”btnbtn-large btn-outline
mx-auto”>Learn about integrations</a>
```

Explanation:
Href has been identified by JAWS/NVDA and reading it out as 'link' along with the text provided as 'Learn about integrations' on the html properties shown earlier.

Gherkin BDD:
For the example of 'Learn about integrations' object, it can be written in a Then statement as follows:

```
Then I tab from “compare price plans”
And I verify “href” of “/html/body/div[4]/div[4]/div/div[1]/p[2]/a” as
“/features/integrations”
```

'href' is the property identified from the html properties of the web page and xpath of 'Learn about integrations' object is '/html/body/div[4]/div[4]/div/div[1]/p[2]/a'; Similarly href of this object is '/features/integrations'; So the test basically navigate to the xpath of the object and get the

href details of the respective object and try to match that with the string '/features/integrations' mentioned in the test. This is known as 'assertions' part of selenium testing.

Step Definition:

```
this.Then(/^I tab from .*\"(.*)\".*/, function(qaselector){
qaselector= this.getSelector(qaselector);
returnthis.client
.isVisible(qaselector)
.keys('Tab')
});

this.Then(/^I verify .*\"(.*)\".* of .*\"(.*)\".* as (.*)/, function(attribute,qaselector,expRes){
qaselector= this.getSelector(qaselector);
returnthis.client
.isVisible(qaselector)
.getAttribute(qaselector, attribute)
.then(function (readValue) {
this.expect(readValue.toString()).to.equal(expRes.toString());
}.bind(this));
});
```

WebdriverIO in JavaScript has been used to construct the framework in this example; WebStorm IDE has been used to write these scripts as part of this NodeJS JavaScript framework.

First step definition just performs Tab keystroke from a link prior to reach 'Learn about integrations' hence the cursor or focus goes from previous link on the page to this particular object.

In the second step definition, xpath of the link 'learn about integrations' is referred through qaselector object and the string passed through this qaselector is /html/body/div[4]/div[4]/div/div[1]/p[2]/a; Once the object has been identified, keyboard keystroke 'Tab' has been pressed by the automated scripts using .keys('Tab') which means that the instructions sent through selenium to browser driver and browser driver perform tab action on the browser and navigate away from 'Learn about integrations.'

When there is a / within double quotes of gherkin BDD script, it will throw errors and it has to be update such as /\. Also, href properties may vary based on the security layer of the organizations. So please find the alternative script (in which you can also perform tab and verification in same step definition):

Gherkin BDD:

```
Then I press tab key to navigate to verify “href” of “/html/body/div[4]/div[4]/div/div[1]/p[2]/a” as “integrations”
```

Step Definition:

```
this.Then(/^I press tab key to navigate to verify .*\”(.*)\”.* of .*\”(.*)\”.* as (.*)/, function(attribute,qaselector,expRes){
qaselector= this.getSelector(qaselector);
returnthis.client
.keys(‘Tab’)
.isVisible(qaselector)
.getAttribute(qaselector, attribute)
.then(function (readValue) {
this.expect(readValue.toString()).to.contains(expRes.toString());
}.bind(this));
});
```

Example script for Down Arrow:

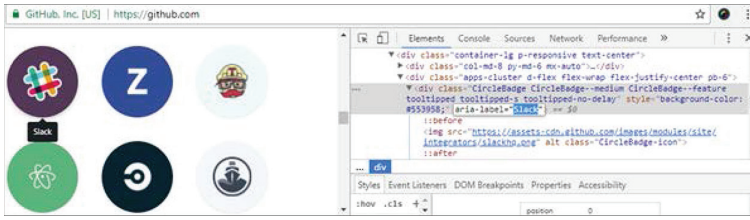

Figure 7.2. Aria label.

Gherkin BDD:

```
Then I press down key to navigate to verify “aria-label” of “/html/body/div[4]/div[4]/div/div[2]/div[1]” as “slack”
```

Step Definition:

```
this.Then(/^I press down key to navigate to verify .*\"(.*)\".* of
.*\"(.*)\".* as (.*)/, function(attribute,qaselector,expRes){
qaselector= this.getSelector(qaselector);
returnthis.client
.keys('ArrowDown')
.isVisible(qaselector)
.getAttribute(qaselector, attribute)
.then(function (readValue) {
this.expect(readValue.toString()).to.equal(expRes.toString());
}.bind(this));
});
```

Reference:

W3c (2018) Keyboard Actions [Web Log Post]. Retrieved from https://w3c.github.io/webdriver/#keyboard-actions

webdriver.io 2018. keys [Web Log Post]. Retrieved from http://webdriver.io/api/protocol/keys.html

Entire keys supported by Selenium 2018. [Web Log Post]. Retrieved from https://w3c.github.io/webdriver/#keyboard-actions

//Xpath of the image slack is /html/body/div[4]/div[4]/div/div[2]/div[1]

Code Explanation

Assume that the cursor or focus is on the link 'learn about integrations' and pressing 'ArrowDown' key from keyboard result in reading out the 'slack' image below the link:

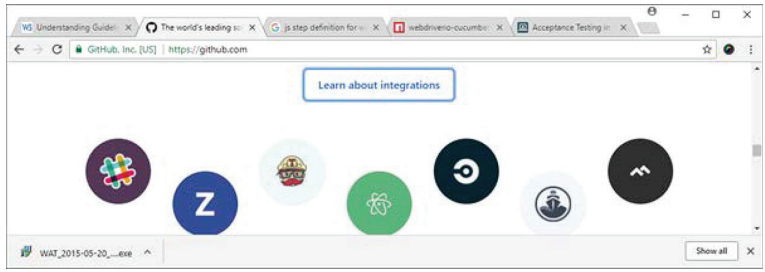

Figure 7.3. Github website.

This is the same behavior expected when using NVDA in Firefox:
Navigate to the link using Tab key:

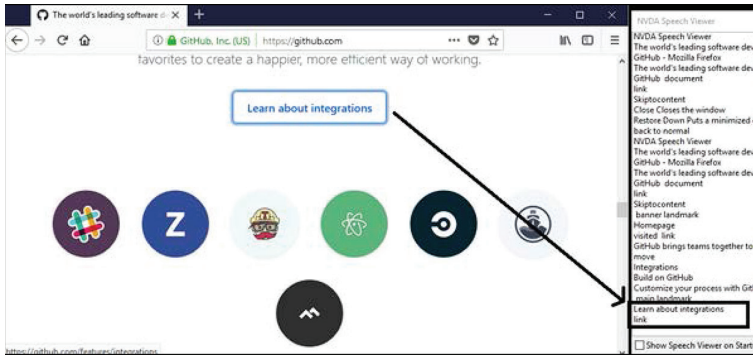

Figure 7.4. A link on Github website.

Press 'ArrowDown' Arrow to observe that the cursor still points to link but the NVDA reads out slack image:

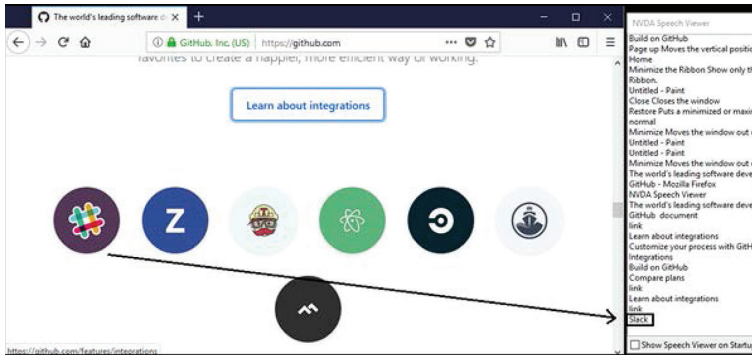

Figure 7.5. Slack image has been read from NVDA screen reader.

This down arrow's behavior is automated using *.keys* ('ArrowDown') function of webdriverio (selenium)

Code example to write down key within Java programming-based Selenium Tests:

```
packageseleniumJavaFramework.KeyboardNavigationTest;
importjava.util.concurrent.TimeUnit;
importorg.openqa.selenium.By;
importorg.testng.annotations.BeforeTest;
importorg.testng.annotations.Test;
importorg.testng.annotations.AfterTest;
```

```
importorg.openqa.selenium.WebDriver;
importorg.openqa.selenium.Keys;
importorg.openqa.selenium.firefox.FirefoxDriver;

public class pressTabTest{
WebDriver driver = new FirefoxDriver();
@BeforeTest
public void setUp(){
driver.get("https://github.com/");
}
@Test
public void findElementbyXpath(){
WebElementqeElement = driver.findElement(By.xpath(".//*[@
id='user[login]']"))
qeElement.click();
qeElement.sendKeys(Keys.DOWN);
}
@AfterTest
public void doThis(){
driver.quit();
}
}
```

What would be a defect look like?
Let us take an example of 'sequence' within this section and test a page for sequence of table items:

Page display: Heading 2 and Heading 4 are displayed within the table and there is a link below the table (outside) such as 'Contact us to learn more' This has been read by NVDA on Firefox and following are the display of speech viewer of NVDA (which has been read to user):

It is expected that the contents within the table has to read out before moving out of the table for proper sequencing. In this case, heading 2 has been read out and then focus goes to link outside the table (when tab has been pressed) and further navigation from the link (contact us to learn more) navigate back to the table and read the heading at level 4 which is a defect (Level AAA) in sequence.

Practice script from GitHub repository
Feature File:

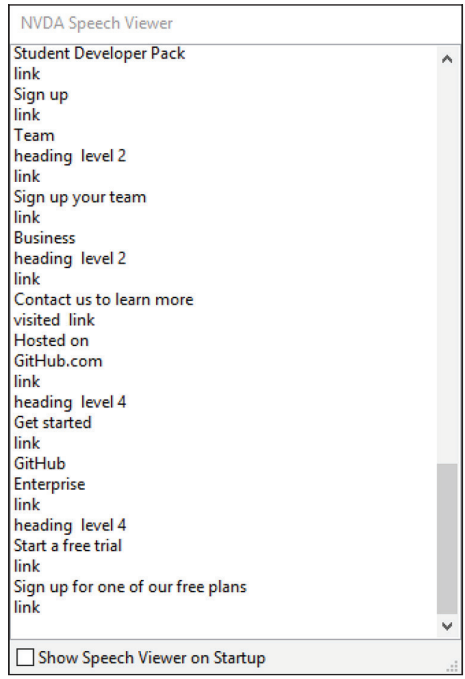

Figure 7.6. NVDA speech viewer.

Feature: Searching for GitHub projects
As an internet user
In order to find out more about GitHub projects
I want to be able to search for information about webdriverio
Scenario: GitHubwebdriverio search
When I search GitHub for "webdriverio"
Then I should see some results
Scenario: 1.3.2 WCAG sequence test-Tab from username to email
When I click on username
Then I press tab key to navigate to verify placeholder of email as you@example.com

Step Definitions:

```
'use strict';

var{defineSupportCode} = require('cucumber');
var{By, until, Keyz} = require('selenium-webdriver');
var{expect} = require('chai');
```

```
defineSupportCode(function({When, Then}) {

When(/^I search GitHub for "([^"]*)"$/, function (searchQuery,
next) {
this.driver.get('https://github.com/');
this.driver.findElement(By.name('q'))
.sendKeys(searchQuery);
this.driver.findElement(By.name('q'))
.sendKeys(Key.ENTER)
.then(function() {
next();
});
});

When(/^I click on username$/, function (next) {
this.driver.get('https://github.com/');
this.driver.findElement(By.id('user[login]' ))
.click()
.then(function() {
next();
});
});

Then(/^I should see some results$/, function (next) {
this.driver.wait(until.elementLocated(By.id('js-pjax-container')));
this.driver.findElements(By.id('js-pjax-container'))
.then(function(elements) {
expect(elements.length).to.not.equal(0);
next();
});
});

Then(/^I press tab key to navigate to verify (.*) of email as (.*)/,
function(attribute,expRes,next){

varelement_to='user[email]';
varelement_from='user[login]';
this.driver.findElement(By.id(element_from))
.sendKeys(Key.TAB);
this.driver.findElement(By.id(element_to)).getAttribute(attribute)
.then(function (readValue) {
```

```
expect(readValue).to.equal(expRes);
next();
});
});




});
```

Test Execution: Right click on cucumber.js and Run (using webstorm IDE tool)

Results:

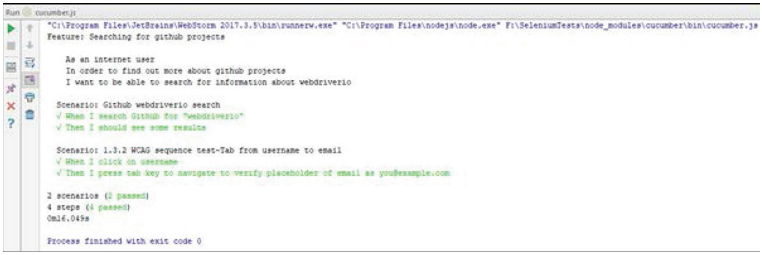

Figure 7.7. Cucumber script test execution status.

CHAPTER 8

TESTING WCAG ACCESSIBILITY STANDARD—DISTINGUISHABLE

When the web pages are provided with multiple colors, it may not be understood by users with medical conditions such as the following.

Some medical conditions related to colors: Color blindness, Achromatopsia, Heterochromia, Synesthesia, Monochromacy, Dichromacy, Tritanopia, Color Monochromacy.

Since special users will not be in a position to access the colors in proper way, it has to be handled fair on the web pages with expected color contrast (so partially blind users can view and understand) and text resize options (so special user can resize text up to 200 percent without using any additional tools), and so on.

Usage of 'Web Accessibility Toolbar'

Web accessibility toolbar (WAT) provide color contrast analyzer when installed and used from internet explorer. This can be used to validate the colors of each object and respective contrast of the colors in normal mode and zoomed mode. As per the WCAG standards, 4.5:1 is the suggested minimum color contrast in normal mode and 3:1 when zoomed.

Installation:

The Paciello Group. March 15, 2018. "Web Accessibility Toolbar." *Web Log Post*. Retrieved from https://developer.paciellogroup.com/resources/wat/

8.1 SAMPLE TEST ON GITHUB.COM FOR CONTRAST VERIFICATION

TestCaseName: 1.4_NFT_Accessibility_IEXWAT_Verify_Github_website
Test Description: Following WCAG standards are verified on the web page www.github.com using Firefox and NVDA

1.4 Distinguishable: Understanding Guideline 1.4
1.4.3 Contrast (Minimum)

8.1.1 REFERENCE

W3C 2016. "Distinguishable: Understanding Guideline 1.4." *Web Log Post*. Retrieved from https://w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast.html
Pre-requisite: Install WAT (open source) and IE on the Windows 10
Test Step 1: Open WAT on an Internet Explorer 11 of Windows 10
Expected Result 1: WAT toolbar should be displayed in IE11
Test Step 2: Launch www.github.com website on IE11 and click on Color Contrast Analyzer tool from WAT
Expected Result 2: Color Contrast of WAT toolbar should be displayed in IE11
Test Step 3: Check the background and foreground color of GitHub logo
Expected Result 3: In Normal mode (100 percent), color contrast has to be above 4.5:1 ratio
Actual Result:
When the page has been zoomed at 300 percent, test passes with color contrast above 3:1 ratio:

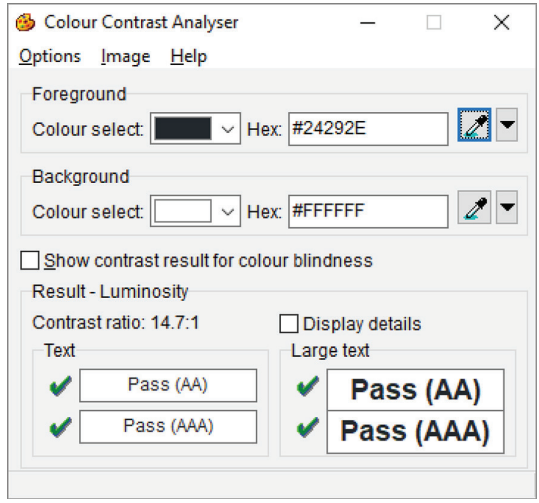

Figure 8.1. Color contrast analyzer.

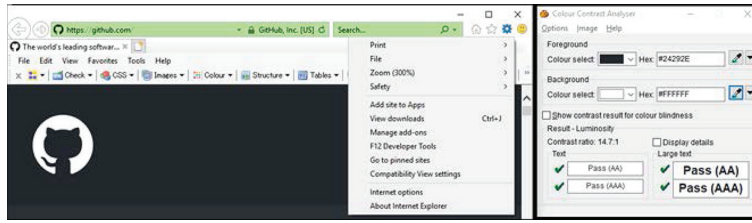

Figure 8.2. Color validation on Github website.

8.2 WRITING AUTOMATION TESTS FOR COLOR CONTRAST ANALYZER USING SELENIUM

WebdriverIO (JavaScript version of selenium) provides a better feature to verify the CSS properties of the objects within the web pages hence the color contrasts can be checked on any given page during regular behavior driven development (BDD) tests.
Example script:
Gherkin BDD:

```
Then I expect to see “colour” of “//*[@class=”octiconocticon-mark
-GitHub”]” as “#fff”
```

Step Definition:

```
this.Then(/^Then I expect to see .*\”(.*)\”.* of .*\”(.*)\”.* as (.*)/,
function(cssElement,qaselector,expRes){
qaselector= this.getSelector(qaselector);
returnthis.client
.isVisible(qaselector)
.getCssProperty(qaselector,cssElement)
.then(function (readValue) {
this.expect(readValue.parsed.hex).to.equal(expRes);
}.bind(this));
});
```

//object property of GitHub logo is //*[@class=”octiconocticon-mark-GitHub”]
Code Explanation
Colour is a CSS property and available to view once inspected from browser and this can be validated from selenium function called ‘getCssProperty’

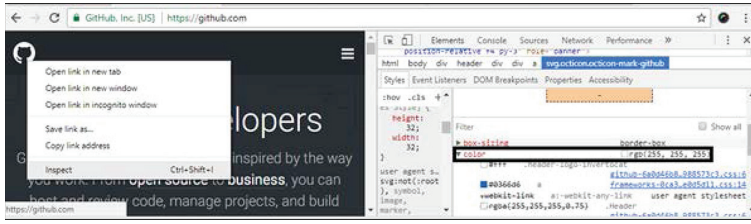

Figure 8.3. Color details of Github website.

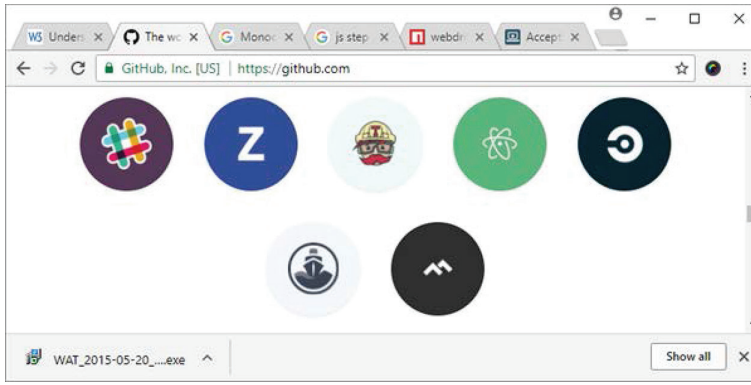

Figure 8.4. Github website.

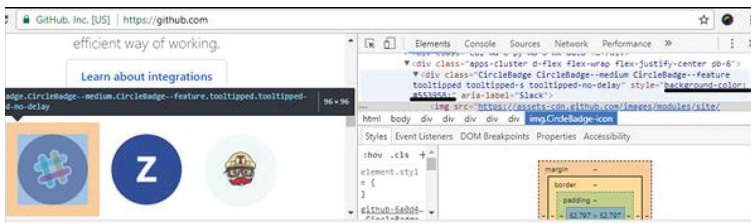

Figure 8.5. Background color details of slack image from Github website.

So test has been updated to check the colour has been reflected with expected result when the page has been displayed.
Similar tests on other images:
Let us take the example of testing first image on the left-hand side:
CSS property(style=”background-colour: #553958;”) of the image:
This can be automated as the BDD tests given as follows:
Gherkin BDD:

```
Then I expect to see “background-colour” of “/html/body/div[4]/
div[4]/div/div[2]/div[1]” as “#553958”
```

Step Definition:

```
this.Then(/^Then I expect to see .*\"(.*)\".* of .*\"(.*)\".* as (.*)/,
function(cssElement,qaselector,expRes){
qaselector= this.getSelector(qaselector);
returnthis.client
.isVisible(qaselector)
.getCssProperty(qaselector,cssElement)
.then(function (readValue) {
this.expect(readValue.parsed.hex).to.equal(expRes);
}.bind(this));
});
```

//Xpath of the image is /html/body/div[4]/div[4]/div/div[2]/div[1]

8.3 SAMPLE TEST ON GITHUB.COM FOR CONTRAST VERIFICATION USING PA11Y

Test Case Name: 1.4_NFT_Accessibility_Pa11y_Verify_Github_website

Test Description: Following WCAG standards are verified on the web page www.github.com using Pa11y

1.4 Distinguishable: Understanding Guideline 1.4

1.4.3 Contrast (Minimum)

8.3.1 REFERENCE

W3C. 2016. “Distinguishable: Understanding Guideline 1.4.” Web Log Post. Retrieved from https://w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast.html

Pre-requisite: Install Pa11y (open source) on existing BDD repository of webdriverIO(selenium) using JavaScript

Test Step 1: Run pa11y for the URL of the web application

Expected Result 1: Reports should be generated

Test Step 2: Discuss with developers and environment engineers to rule out ‘false positives’ described in pa11y reports and identify valid issues from the report

Expected Result 2: Valid defects should be handpicked and reported to the agile team to update the backlog and fix the items

Configuring Pa11y into BDD:

Update package.json file with pa11y versions:

```
{
“name”: “seleniumtests”,
“version”: “1.0.0”,
“description”: “automated accessibility tests using selenium”,
“main”: “index.js”,
“scripts”: {
“test”: “node_modules/cucumber/bin/cucumber.js”
},
“author”: “Narayanan Palani”,
“license”: “MIT”,
“devDependencies”: {
“chai”: “^4.1.2”,
“cucumber”: “^2.3.1”,
“pa11y”: “^4.13.2”,
“pa11y-reporter-1.0-json”: “^1.0.2”,
“sanitize-filename”: “^1.6.1”,
“selenium-webdriver”: “^3.5.0”
}
}
```

Perform installation through following commands for windows:

```
npm install -g pa11y
```

Once installation completed, perform few tests to understand how it works? Go to terminal(of WebStorm IDE) and type:

F:\SeleniumTests>**pa11y https://www.github.com**

This would get a report in the following format:

```
Welcome to Pa11y

> Running Pa11y on URL https://www.github.com

Results for URL: https://github.com/

• Error: This form does not contain a submit button, which creates issues for those who cannot submit the form using the keyboard. Submit buttons are INPUT elements with type attribute
“submit” or “image”, or BUTTON elements with type “submit” or omitted/invalid.
```

├── WCAG2AA.Principle3.Guideline3_2.3_2_2.H32.2
├── html> body >div:nth-child(1) > header > div >div:nth-child(2) > div > div > div > form
└──<form class="js-site-search-form" data-unscoped-search-URL="/search" action="/search" accept-charset="UTF-8" method="get"><input name="utf8" type="hidden...</form>

• Error: This element has insufficient contrast at this conformance level. Expected a contrast ratio of at least 4.5:1, but text in this element has a contrast ratio of 2.3:1. Recommendation: change text colour to #fff.
├── WCAG2AA.Principle1.Guideline1_4.1_4_3.G18.Fail
├── html> body >div:nth-child(1) > header > div >div:nth-child(2) > div > span > div > span
└── or

• Error: This text input element does not have a name available to an accessibility API. Valid names are: label element, title attribute, aria-label attribute, aria-labelledby attribute.
├── WCAG2AA.Principle4.Guideline4_1.4_1_2.H91.InputText.Name
├── #required_field_6a4c
└── <input type="text" name="required_field_6a4c" id="required_field_6a4c" style="display: none" class="form-control">

• Error: This form field should be labeled in some way. Use the label element (either with a "for" attribute or wrapped around the form field), or "title", "aria-label" or "aria-labell
edby" attributes as appropriate.
├── WCAG2AA.Principle1.Guideline1_3.1_3_1.F68
├── #required_field_6a4c
└── <input type="text" name="required_field_6a4c" id="required_field_6a4c" style="display: none" class="form-control">

• Error: This element has insufficient contrast at this conformance level. Expected a contrast ratio of at least 4.5:1, but text in this element has a contrast ratio of 3.49:1. Recommendation: change text colour to #c85200.
├── WCAG2AA.Principle1.Guideline1_4.1_4_3.G18.Fail
├── html > body >div:nth-child(4) >div:nth-child(5) >div:nth-child(3) >div:nth-child(2) > div > a:nth-child(1) >div:nth-child(2) > p > span
└──open source community

• Error: This text input element does not have a name available to an accessibility API. Valid names are: label element, title attribute, aria-label attribute, aria-labelled by attribute.
├── WCAG2AA.Principle4.Guideline4_1.4_1_2.H91.InputText.Name
├── #required_field_ae94
└── <input type=”text” name=”required_field_ae94” id=”required_field_ae94” style=”display: none” class=”form-control”>

• Error: This form field should be labeled in some way. Use the label element (either with a “for” attribute or wrapped around the form field), or “title”, “aria-label” or “aria-labell
edby” attributes as appropriate.
├── WCAG2AA.Principle1.Guideline1_3.1_3_1.F68
├── #required_field_ae94
└── <input type=”text” name=”required_field_ae94” id=”required_field_ae94” style=”display: none” class=”form-control”>

• Error: This element has insufficient contrast at this conformance level. Expected a contrast ratio of at least 4.5:1, but text in this element has a contrast ratio of 3.05:1. Recommendation: change text colour to #fff.
├── WCAG2AA.Principle1.Guideline1_4.1_4_3.G18.Fail
├── html> body >div:nth-child(4) >div:nth-child(6) >div:nth-child(2) > form > p
└── <p class=”form-control-note text-center mt-6”>By clicking “Sign up...</p>

• Error: This element has insufficient contrast at this conformance level. Expected a contrast ratio of at least 4.5:1, but text in this element has a contrast ratio of 3.05:1. Recommendation: change text colour to #fff.
├── WCAG2AA.Principle1.Guideline1_4.1_4_3.G18.Fail
├── html> body >div:nth-child(4) >div:nth-child(6) >div:nth-child(2) > form > p > span
└── We’ll occasionally send you acc...

9 Errors

8.4 JUSTIFICATION

This means that there are nine errors on the page related Accessibility AA standards WCAG2 since pa11y scan the pages on AA standards by default. If we need a report for particular standard such as WCAG2's A or AAA or Section 508, then it has to be specified on the command itself.

8.5 FURTHER INVESTIGATION

Following Error can be verified on the web page by using the xpath provided on the error report

• Error: This element has insufficient contrast at this conformance level. Expected a contrast ratio of at least 4.5:1, but text in this element has a contrast ratio of 2.3:1. Recommendation: change text colour to #fff.
├── WCAG2AA.Principle1.Guideline1_4.1_4_3.G18.Fail
├── html> body >div:nth-child(1) > header > div >div:nth-child(2) > div > span > div > span
└── or

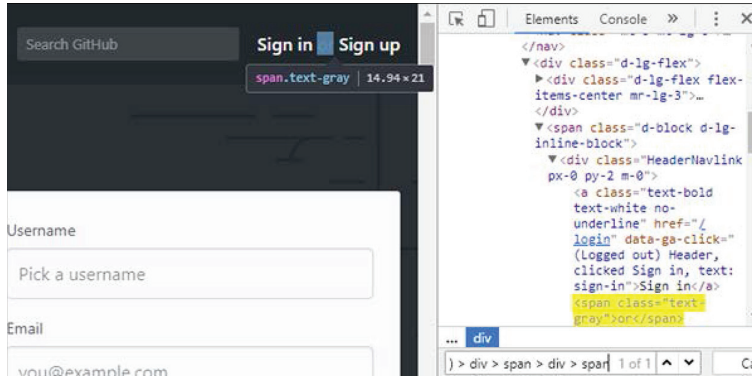

Figure 8.6. Class details of the text from Github.

Copy the xpath and search that in Elements section on Google Chrome for the web page (Figure 8.6).
Why Pa11y reports this 'or' text as a defect on Guideline1_4.1_4_3.G18?

Minimum requirement of having contrast ratio of 4.5:1 for the object has been described on the following sections

W3C 2016. “Contrast.” *Web Log Post*. Retrieved from https://w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast.html
W3C 2016. “Contrast.” *Web Log Post*. Retrieved from https://w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html

Open the website www.github.com in Internet Explorer and verify the colour contrast using WAT tool’s colour contrast analyzer and realize the ‘or’ text failing in contrast ratio by having met only 2.3:1 ratio when 4.5:1 is required as a minimum contrast ratio (Figure 8.7).

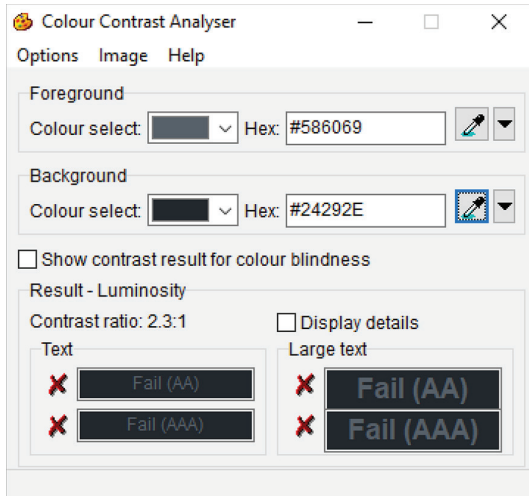

Figure 8.7. Color contrast analyzer shows a test failure.

Hence one of the errors provided in Pa11y report is valid and similarly other errors are to be verified to check if there is any fix required.
Extract report to external file:
AA Classification Pa11y Issues:

When standard is not mentioned in the command, pa11y scans the page for AA WCAG standards by default as follows:

F:\SeleniumTests>**pa11y --reporter csv https://www.github.com >accessibilityreportv1.csv**

A Classification Pa11y Issues:

F:\SeleniumTests>**pa11y --reporter csv --standard WCAG2A https://www.github.com >accessibilityreportWCAG2A.csv**

Report from the csv file:

type	Code	Message	Context	Selector	
error	WCAG2A.Principle3.Guideline3_2.3_2_2.H32.2	This form does not contain a submit button, which creates issues for those who cannot submit the form using the keyboard. Submit buttons are INPUT elements with type attribute \ submit\" or \"image\"	or BUTTON elements with type \"submit\" or omitted/invalid."	<form class=\js-site-search-form\" data-unscoped-search-URL=\"/search\" action=\"/search\" accept-charset=\"UTF-8\" method=\"get\"><input name=\"utf8\" type=\"hidden...</form>"	html > body >div:nth-child(1) > header > div >div:nth-child(2) > div > div > div > form
error	WCAG2A.Principle4.Guideline4_1.4_1_2.H91.InputText.Name	This text input element does not have a name available to an accessibility API. Valid names are: label element, title attribute, aria-label attribute, aria-labelledby attribute.	<input type=\text\" name=\"required_field_6959\" id=\"required_field_6959\" style=\"display: none\" class=\"form-control\">"	#required_field_6959	

(Continued)

type	Code	Message	Context	Selector		
error	WCAG2A.Principle1.Guideline1_3.1_3_1.F68	This form field should be labelled in some way. Use the label element (either with a \for\" attribute or wrapped around the form field)	or \"title\"	\"aria-label\" or \"aria-labeledby\" attributes as appropriate."	<input type=\text\" name=\"required_field_6959\" id=\"required_field_6959\" style=\"display: none\" class=\"-form-control\">"	#required_field_6959
error	WCAG2A.Principle4.Guideline4_1.4_1_2.H91.InputText.Name	This text input element does not have a name available to an accessibility API. Valid names are: label element, title attribute, aria-label attribute, aria-labelledby attribute.	<input type=\text\" name=\"required_field_9b93\" id=\"required_field_9b93\" style=\"display: none\" class=\"form-control\">"	#required_field_9b93		

error	WCAG2A.Principle1.Guideline1_3.1_3_1.F68	This form field should be labeled in some way. Use the label element (either with a \for\" attribute or wrapped around the form field)	or \"title\"	\"aria-label\" or \"aria-labelledby\" attributes as appropriate."	<input type=\text\" name=\"required_field_9b93\" id=\"required_field_9b93\" style=\"display: none\" class=\"-form-control\">"	#required_field_9b93

type	Code	Message	Context	Selector		
error	Section508.L.NoContentAnchor	Anchor element found with a valid href attribute, but no link content has been supplied.	<a class=\header-logo-invertocat my-0\" href=\"https://github.com/\" aria-label=\"Homepage\" data-ga-click=\"(Logged out) Header	go to homepage	icon:logo-wordmark\">\n<svg height=\"32\" class..."	html > body >div:nth-child(1) > header > div >div:nth-child(1) > a
error	Section508.D.HeadingOrder	The heading structure is not logically nested. This h6 element should be an h2 to be properly nested.	<h6 class=\alt-h4 text-normal text-gray text-center\">\nGitHub for teams\n</h6>"	html > body >div:nth-child(4) >div:nth-child(2) >div:nth-child(1) > h6		
error	Section508.D.HeadingOrder	The heading structure is not logically nested. This h6 element should be an h4 to be properly nested.	<h6 class=\alt-h4 text-normal text-gray text-center\">\nSecurity and administrat...</h6>"	html > body >div:nth-child(4) >div:nth-child(3) > div > h6		

error	Section508.D.HeadingOrder	The heading structure is not logically nested. This h6 element should be an h5 to be properly nested.	<h6 class=\alt-h4 text-normal text-gray\”>\nIntegrations\n</h6>”	html > body >div:nth-child(4) >div:nth-child(4) > div >div:nth-child(1) > h6
error	Section508.D.HeadingOrder	The heading structure is not logically nested. This h6 element should be an h3 to be properly nested.	<h6 class=\alt-h4 text-normal text-gray text-center\”>\nCommunity\n</h6>”	html > body >div:nth-child(4) >div:nth-child(5) >div:nth-child(1) > h6
error	Section508.D.HeadingOrder	The heading structure is not logically nested. This h3 element should be an h2 to be properly nested.	<h3 class=\alt-h3 text-gray-light mt-3 mb-4 lh-condensed\”>\nMore than 1.8 million<...</h3>”	html > body >div:nth-child(4) >div:nth-child(5) >div:nth-child(3) >div:nth-child(3) > h3
error	Section508.D.HeadingOrder	The heading structure is not logically nested. This h4 element should be an h3 to be properly nested.	<h4 class=\mb-2\”>-Features</h4>”	html > body >div:nth-child(5) > div >div:nth-child(2) > h4

Pa11y Issues Related to Section508 Guidelines:

F:\SeleniumTests>**pa11y --reporter csv --standard Section508**

https://www.github.com >accessibilityreportS508.csv

Report from the csv file:
AAA Classification Pa11y Issues:

F:\SeleniumTests>**pa11y --reporter csv --standard WCAG2AAA**

https://github.com >accessibilityreportWCAG2AAA.csv

8.6 SAMPLE TEST ON GITHUB.COM FOR 1.4.2 AUDIO CONTROL

Test Case Name: 1.4.2_NFT_Accessibility_FFXNVDA_Verify_Github_website

Test Description: Following WCAG standards are verified on the web page www.github.com using Firefox and NVDA

1.4 Distinguishable: Understanding Guideline 1.4

1.4.2 Audio Control

8.6.1 REFERENCE

W3C 2016. "Distinguishable: Understanding Guideline 1.4." *Web Log Post*. Retrieved from https://w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast.html

Pre-requisite: Install NVDA (open source) and Firefox quantum 58.0.2 on the Windows 10

Test Step 1: Open NVDA on a Windows 10 operating system and use earphone to hear the voice provided by NVDA

Expected Result 1: NVDA should be opened successfully and contents read out clearly by NVDA to user.

Test Step 2: Open Firefox Quantum (version 58.0.2 (64-bit)) on a Windows 10 operating system

Expected Result 2: Firefox should be opened successfully.

Test Step 3: Launch www.github.com/features website

Expected Result 3: GitHub page should be opened and URL should be read successfully by NVDA

Test Step 4: Navigate from URL to "New to GitHub? See how it works" link using TAB key of keyboard

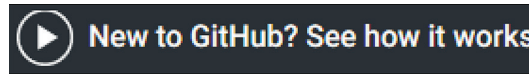

Figure 8.8. Github video.

Expected Result 4: NVDA should read the details of the link clearly
Test Step 5: Press Enter from Keyboard
Expected Result 5: User should be navigated to listen to video and audio of the GitHub YouTube and user should be allowed to pause, change settings or stop the video to come back to the page where the link has been clicked
Actual Result:
Clicking/Pressing Enter on the link 'New to GitHub? See how it works' leads to display video along with audio and user 'Tab' navigations help to pause or change settings or stop the video to come back to the parent page and pressing Tab from there is leading to navigation to next sections as expected.

CHAPTER 9

TESTING WCAG ACCESSIBILITY STANDARD—UNDERSTANDABLE

While testing web applications, if testers find complex content which are hard to read, they should raise this to Business Analysts part of accessibility testing. Developers should avoid complex texts and make it simple to read and understand the content on the web page. If the content is complex in nature and hard to read and understand the purpose of it, it may not be considered as defect since developers develop the page exactly how it has been instructed from user stories or requirements documents. In such cases, these clarifications can be raised as 'Issue' or 'Recommendations' of defect tracking system to get clear instructions on how to make it better.

9.1 EXAMPLE: INPUT ASSISTANCE ON A LOGIN PAGE

If a user is trying to login on a web page with user id and password, error messages and instructions should be displayed to users at right time. If user entered user id and left the password text box blank, the error message should be appropriate and suggest user to fill password to continue login. Similarly, pressing TAB key using keyboard should assist users to navigate from user id text box to password text box navigations. If user want to navigate backwards using 'Shift+Tab' key, there should be clear navigation instruction provided to users through audio. Pressing 'Shift+Tab' from password field, should reach 'user id' field and speech assistance should read details about user id field to user. If user entered invalid user id and password and clicked on login button should be provided with valid error message to re-enter valid user id and password details. When user is

provided with error message on screen but failed to read out using speech over, it is an accessibility defect since users with assistive technology such as JAWS or NVDA will not know what kind of error has occurred during the login process. Following test case is one good example to validate this login process:

1.19.1_NFT_Accessibility_FFXNVDA_Verify_School_Student_Assingment_website_login

Test Description: Following WCAG standards are verified on the web page using Firefox and NVDA while login to the web page.

3.3.3 Error Suggestion

9.1.1 REFERENCE

W3C 2016. "Input Assistance: Understanding Guideline 3.3." *Web Log Post*. Retrieved from https://w3.org/TR/UNDERSTANDING-WCAG20/minimize-error-suggestions.html

Pre-requisite: Install NVDA (open source) and Firefox quantum 58.0.2 on the Windows 10

Test Step 1: Open NVDA on a Windows 10 operating system and use earphone to hear the voice provided by NVDA

Expected Result 1: NVDA should be opened successfully and contents read out clearly by NVDA to user.

Test Step 2: Open Firefox Quantum (version 58.0.2 (64-bit)) on a Windows 10 operating system

Expected Result 2: Firefox should be opened successfully.

Test Step 3: Launch Student Assignment website login portal

Expected Result 3: Students login page should be opened and URL should be read successfully by NVDA

Test Step 4: Press TAB key to navigate to user id field to enter valid user id and press TAB key again to enter invalid password and enter another TAB key to navigate to Login button and press Enter

Expected Result 4: NVDA should read out about the same error message displayed on the screen which should be understandable and valid error message to correct the password entered.

CHAPTER 10

Testing WCAG Accessibility Standard—Robust

When readers reading a content from web application using JAWS, the content should be reliable enough to provide details of sections and reader should have similar experience when using NVDA and mobile device-voice over mode!

Case study: Hema is 15 years old and affected by Dyslexia, common learning difficulty with reading, writing and spelling. She login to her school online account to access assignments and read the assignments list using NVDA audio and 'speech viewer' options. While reading the assignments section, she pressed 'INSERT+F7' keyboard actions to see listed links of entire page in which assignments listed as links and she navigate to each link where she needs to attend them and complete for a particular week. The student online application has undergone software upgrade and visually no change to the assignments page. It is clearly visible when using from computer but 'Listed Links' from NVDA display and read URLs of link rather than providing details about the link in text format. If she accesses the same page from mobile, voice over of mobile reads the links clearly from mobile view. But she could not download relevant assignments from mobile hence she depends on laptop device to download and access her assignments. What is the exact defect on the application or is there an accessibility defect on the page or not?

Defect: Each link of assignments within assignments page are not updated with 'title' within HTML properties. Every link on web page are provided with 'href' which acts as a link to target page to

proceed on opening the assignments. If html has been provided with 'title' for each link, it displays that relevant information of title part of 'Listed Links' whereas links developed without 'title' result in displaying just URLs from 'href' of html properties hence the bug is on html DOM properties of the student web application.

Suggested Fix: Add 'title' for each link of the web page which should be unique and consistent for each link. If there are two assignments with same name, 'title' of each assignment should be unique, meaningful such as 'assignment 1' and 'assignment 2', and so on.

10.1 IMPORTANCE OF NAME, ROLE, VALUE ON PUBLIC WEBSITES

All form elements, text boxes, links, buttons, images and other components of the web page should be available with name, roles (such as button, link or text box, and so on) and value within html properties.

Example:
Updating html properties such as title, aria-label would help in providing best experience while accessing the page with assistive technologies such as JAWS, NVDA, and so on.

More examples are available at: https://w3.org/TR/WCAG20-TECHS/ARIA16.html

Sample Test Case on Student Assignment Webpage for WCAG-Robust Guidelines

10.2 TEST CASE FOR NVDA

1.19.1_NFT_Accessibility_FFXNVDA_Verify_School_Student_Assingment_website

Test Description: Following WCAG standards are verified on the web page using Firefox and NVDA

4.1.2 Name Role Value

10.2.1 REFERENCE

W3C 2016. "Name, Role, Value: Understanding SC 4.1.2." *Web Log Post*. Retrieved from https://w3.org/WAI/WCAG21/quickref/?versions=2.0&-showtechniques=412#name-role-value

Pre-requisite: Install NVDA (open source) and Firefox quantum 58.0.2 on the Windows 10

Test Step 1: Open NVDA on a Windows 10 operating system and use earphone to hear the voice provided by NVDA

Expected Result 1: NVDA should be opened successfully and contents read out clearly by NVDA to user.

Test Step 2: Open Firefox Quantum (version 58.0.2 (64-bit)) on a Windows 10 operating system

Expected Result 2: Firefox should be opened successfully.

Test Step 3: Launch Student Assignment website

Expected Result 3: Students page should be opened and URL should be read successfully by NVDA.

Test Step 4: Navigate from URL to assignments tab using TAB key and press ENTER of keyboard on the students' main page

Expected Result 4: Navigated to Assignments page and NVDA should read the details of the links, text boxes and any images should be read with the text alternatives.

Test Step 5: Press INSERT+F7 key to navigate to the listed links

Expected Result 5: Listed Links should be displayed with complete links available on the page with details about each link with clear information.

10.3 TEST CASE FOR JAWS

1.19.2_NFT_Accessibility_IEXJAWS_Verify_School_Student_Assignment_website

Test Description: Following WCAG standards are verified on the web page using Internet Explorer11 or Microsoft Edge and JAWS

4.1.2 Name Role Value

10.3.1 REFERENCE

W3C 2016. "Name, Role, Value: Understanding SC 4.1.2." *Web Log Post*. Retrieved from https://w3.org/WAI/WCAG21/quickref/?versions=2.0&-showtechniques=412#name-role-value

Pre-requisite: Install JAWS (licensed) and Internet Explorer11 or Microsoft Edge on the Windows 10

Test Step 1: Open JAWS on a Windows 10 operating system and use earphone to hear the voice provided by JAWS

Expected Result 1: JAWS should be opened successfully and contents read out clearly by JAWS to user.

Test Step 2: Open Internet Explorer11 or Microsoft Edge on a Windows 10 operating system

Expected Result 2: Internet Explorer11 or Microsoft Edge should be opened successfully.

Test Step 3: Launch Student Assignment website

Expected Result 3: Students page should be opened and URL should be read successfully by JAWS

Test Step 4: Navigate from URL to assignments tab using TAB key and press ENTER of keyboard on the students' main page

Expected Result 4: Navigated to Assignments page and JAWS should read the details of the links, text boxes and any images should be read with the text alternatives

Test Step 5: Press INSERT+F7 key to navigate to the listed links

Expected Result 5: Listed Links should be displayed with complete links available on the page with details about each link with clear information.

CHAPTER 11

TESTING WCAG ACCESSIBILITY STANDARD—OPERABLE

Special users access the web pages using keyboard shortcuts and most of the navigation are performed with keyboards and mouse access has been extremely minimal. By keeping keyboard in mind, user navigation of the web pages should be verified to avoid any keyboard traps hence Tab, Shift Tab or arrow keys flow in proper sequence for better user experience.

Sample 'A level' (High Priority) defect on Keyboard Trap:

User press TAB from URL to navigate to header 1 and pressing TAB from header 1 goes back to URL rather than navigating to other sections of the page! So, user cannot proceed on any other operation until they get access to mouse or closing the application by switching off computer.

11.1 SAMPLE TEST ON GITHUB.COM FOR 2.1 KEYBOARD ACCESSIBLE

Test Case Name: 2.1_NFT_Accessibility_FFXNVDA_Verify_Github_website

Test Description: Following WCAG standards are verified on the web page www.github.com using Firefox and NVDA

2.0 Operable

2.1 Keyboard Accessible

11.1.1 REFERENCE

W3C 2016. "Operable 2.0." *Web Log Post*. Retrieved from https://w3.org/TR/WCAG21/#operable

Pre-requisite: Install NVDA (open source) and Firefox quantum 58.0.2 on the Windows 10

Test Step 1: Open NVDA on a Windows 10 operating system and use earphone to hear the voice provided by NVDA

Expected Result 1: NVDA should be opened successfully and contents read out clearly by NVDA to user.

Test Step 2: Open Firefox Quantum (version 58.0.2 (64-bit)) on a Windows 10 operating system

Expected Result 2: Firefox should be opened successfully.

Test Step 3: Launch www.github.com website

Expected Result 3: GitHub page should be opened and URL should be read successfully by NVDA.

Test Step 4: Press Tab to navigate to every section/link of the page and move backwards using Shift Tab similarly

Expected Result 4: NVDA should read the details of the links in sequence when Tab has been pressed. Similarly pressing Shift Tab should provide the same sequence of objects when moving backwards.

11.2 TESTING ANIMATIONS ON THE WEB PAGES

As per WCAG 2.2.2 Pause, Stop, Hide, user should be in a position to understand and act on the animations within web pages.

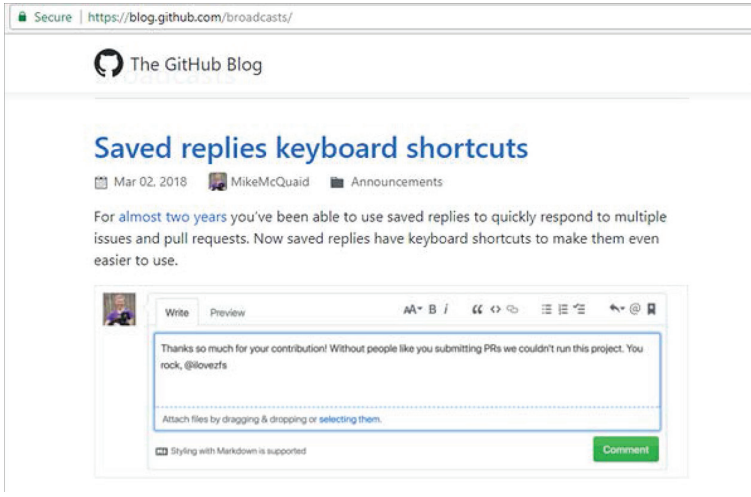

Figure 11.1. Github blog.

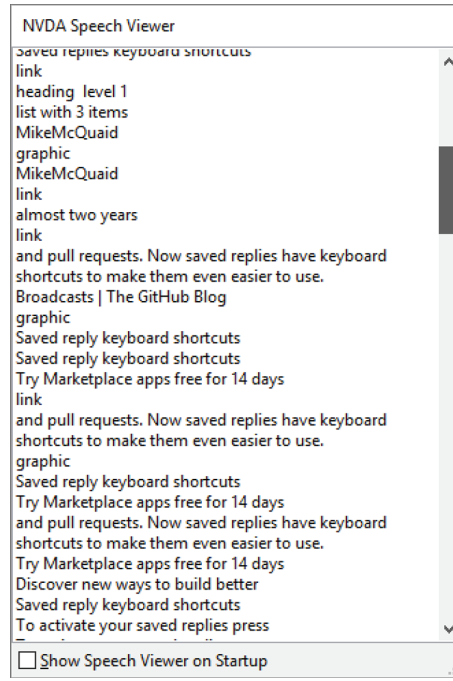

Figure 11.2. NVDA speech viewer.

When there is a small piece of animation loaded as part of the page, this has to be handled fair on the page with a proper text alternative when user navigated to the image.

In the previous picture (Figure 11.1), 'saved reply' animated image has been presented with a text alternative which is captured by NVDA when tested.

NVDA provide the graphic content's text alternative:

Graphic -Saved reply keyboard shortcuts have been provided part of the speech from software

Observations:

When magnifier used to zoom the animation for 300 percent, it is not clear to read some texts presented within animation; As an alternative, page zoom toward 300 percent resolving the clarity issue and the display is clear.

Link purpose verification

The same example given earlier, satisfies the needs of 2.4.4 Link Purpose of WCAG Guidelines since the links are provided with text and the next word specified it as a 'link' for the listeners.

CHAPTER 12

VISUAL REGRESSION TESTS USING BACKSTOPJS

Images, texts, tables and alignment of sections on webpages needs to be displayed with high quality foreground and background colors in order to be viewed by low vision viewers!

Especially diabetic retinopathy patients (complication of diabetic) often have problems in back of their eye and try to use magnifiers to see particular section of the web pages.

12.1 WHY PERFORMING 'VISUAL REGRESSION' IS A SOLUTION FOR TESTING ACCESSIBILITY?

Fixed width designs lead into issues when web browser viewed in different sizes. Especially testing the web page in different view ports (height, width) help to understand how the page displays when reduced the size to mobile view, tablet view and desktop views. If there are fixed width designs, shrinking the page to small mobile view leading the users to use 'horizontal bar' which is extremely difficult for users with low vision (to find horizontal bar and drag it to see remaining part of the web pages); Hence performing visual regression tests will help in reviewing web pages in different sizes and validate the flexibility of the contents!

Often viewers with partial eye visibility use low vision aids such as magnifying spectacles, hand magnifiers, stand magnifiers, telescopes or electronic magnifiers. Hence web pages should be visible with clear contents when viewed large or small in various sizes. Testing webpages with tools such as BackstopJS will be helpful in verification of page contents across various sizes of length and width of the testable web pages.

Pre-requisites:

- Highly recommended to use a selenium-based JavaScript framework such as, https://github.com/john-doherty/selenium-cucumber-js; After cloning this framework to local machine, please install npm modules using commands from GitHub page
- Once framework has been cloned and installed, update package.json with latest backstopjs version such as 3.8.5; Follow further instructions at : https://github.com/garris/BackstopJS
- Perform npm install -g backstopjs to install BackstopJS to the repository

12.2 CREATE CONFIGURATION FILE FOR BACKSTOPJS

Page sizes of the website can be described in following a JSON file for BackstopJS. In order to get JSON file creation, backstop init command has to be performed on terminal:

After backstop init command, following json file should have been created on the main project level.

Note: If Json file has not been created, please rectify any issues with backstopjs installation. If installation is successful but the Json file is not created using backstop init command, please navigate to node modules folder and find backstopjs folder to take a copy of backstop.json file and paste it under the main project folder.

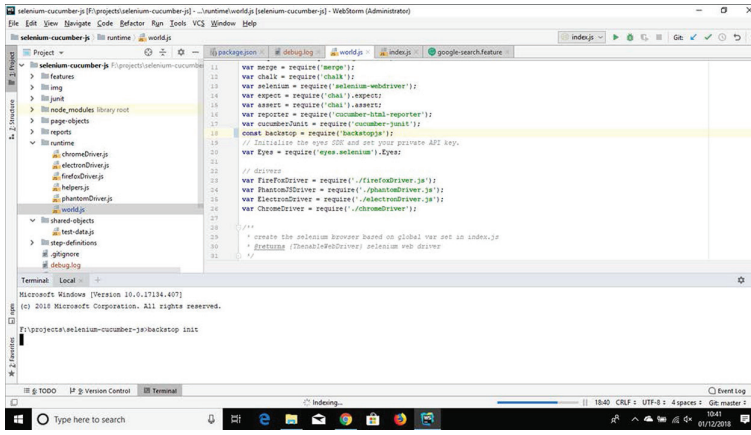

Figure 12.1. Github project.

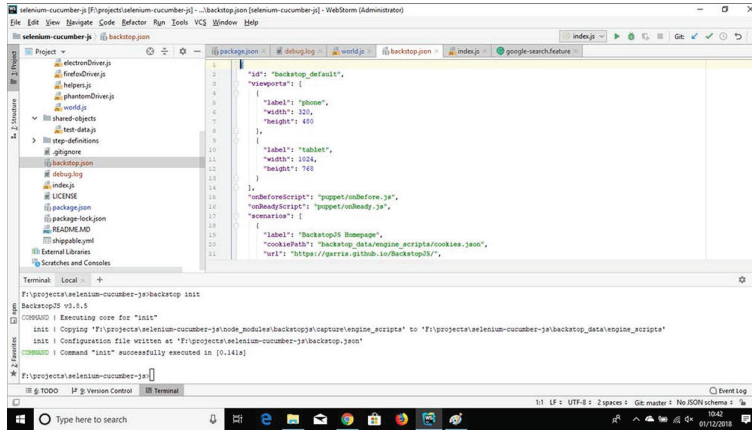

Figure 12.2. Backstop.json file.

Figure 12.3. World.js file.

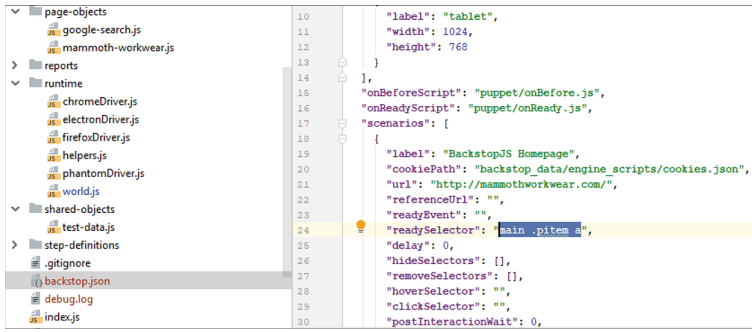

Figure 12.4. Backstop.json file.

Update world configuration of the framework with backstopjs constant declaration.

Once JSON has been made available after backstop init command, please update the config file with relevant objects of the web page.

Please do update the page sizes at various level such as mobile view, tablet view, large and small, and so on.
Example: Sample configuration to run the tests from puppeteer.

```
{
“id”: “backstop_default”,
“viewports”: [
{
“label”: “large”,
“height”: 2400,
“width”: 1024
},
{
“label”: “small”,
“height”: 1000,
“width”: 550
},
{
“label”: “phone”,
“width”: 320,
“height”: 480
},
{
“label”: “tablet”,
“width”: 1024,
“height”: 768
}
],
“onBeforeScript”: “puppet/onBefore.js”,
“onReadyScript”: “puppet/onReady.js”,
“scenarios”: [
{
“label”: “BackstopJS Homepage”,
“cookiePath”: “backstop_data/engine_scripts/cookies.json”,
“URL”: “http://mammothworkwear.com/”,
“referenceUrl”: “”,
“readyEvent”: “”,
“readySelector”: “main .pitem a”,
“delay”: 0,
```

```
“hideSelectors”: [],
“removeSelectors”: [],
“hoverSelector”: “”,
“clickSelector”: “”,
“postInteractionWait”: 0,
“selectors”: [],
“selectorExpansion”: true,
“expect”: 0,
“misMatchThreshold” : 0.1,
“requireSameDimensions”: true
}
],
“paths”: {
“bitmaps_reference”: “backstop_data/bitmaps_reference”,
“bitmaps_test”: “backstop_data/bitmaps_test”,
“engine_scripts”: “backstop_data/engine_scripts”,
“html_report”: “backstop_data/html_report”,
“ci_report”: “backstop_data/ci_report”
},
“report”: [“browser”],
“engine”: “puppeteer”,
“engineOptions”: {
“args”: [“--no-sandbox”]
},
“asyncCaptureLimit”: 5,
“asyncCompareLimit”: 50,
“debug”: false,
“debugWindow”: false
}
```

12.3 TEST EXECUTION USING BACKSTOPJS

Perform ‘backstop test’ command to run the first test which is going to fail since there is no base reference file to compare the images:

If images are okay and recommended to be validated against every tests as a reference image, please perform ‘backstop approve’ command hence every test after this will be comparing against this particular set of files and pass or fail based on comparison. If requirements changes, perform test again and approve right file images.

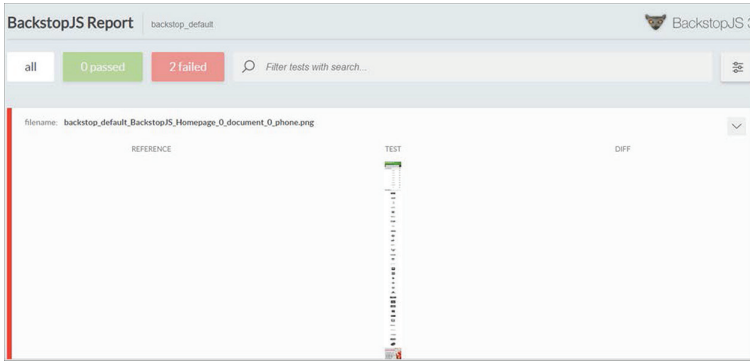

Figure 12.5. BackstopJS report.

```
F:\projects\selenium-cucumber-js>backstop approve
BackstopJS v3.8.5
Loading config:  F:\projects\selenium-cucumber-js\backstop.json

COMMAND | Executing core for "approve"
Copying from backstop_data/bitmaps_test to backstop_data/bitmaps_reference.
The following files will be promoted to reference...
>  backstop_default_BackstopJS_Homepage_0_document_0_large.png
>  backstop_default_BackstopJS_Homepage_0_document_1_small.png
>  backstop_default_BackstopJS_Homepage_0_document_2_phone.png
>  backstop_default_BackstopJS_Homepage_0_document_3_tablet.png
COMMAND | Command "approve" successfully executed in [0.218s]

F:\projects\selenium-cucumber-js>backstop test
BackstopJS v3.8.5
Loading config:  F:\projects\selenium-cucumber-js\backstop.json
```

Figure 12.6. Backstop approve command.

```
Terminal:  Local  +
     compare | OK: BackstopJS Homepage backstop_default_BackstopJS_Homepage_0_document_2_phone.png
     compare | OK: BackstopJS Homepage backstop_default_BackstopJS_Homepage_0_document_1_small.png
     compare | OK: BackstopJS Homepage backstop_default_BackstopJS_Homepage_0_document_0_large.png
     compare | OK: BackstopJS Homepage backstop_default_BackstopJS_Homepage_0_document_3_tablet.png
      report | Test completed...
      report | 4 Passed
      report | 0 Failed
      report | Writing browser report
      report | Resources copied
      report | Copied configuration to: F:\projects\selenium-cucumber-js\backstop_data\html_report\config.js
     COMMAND | Executing core for "openReport"
  openReport | Attempting to ping
  openReport | Remote not found. Opening backstop_data\html_report\index.html
     COMMAND | Command "report" successfully executed in [2.344s]
     COMMAND | Command "test" successfully executed in [13.657s]

F:\projects\selenium-cucumber-js>

 6: TODO   9: Version Control   Terminal
```

Figure 12.7. Test result.

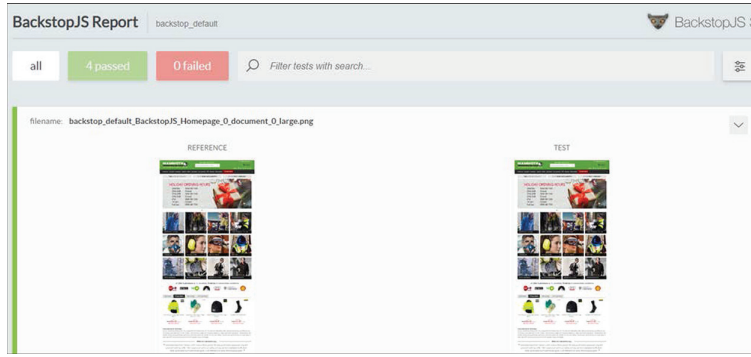

Figure 12.8. BackstopJS report.

Perform backstop test every time to see the comparison of the files and find defects when page is misaligned.

Verify pass or fail through backstop report as follows:
Note: Similar Visual Regression Tests Using Selenium WebdriverIO

Webdriver Visual Regression Service is one of the wonderful tool in performing visual regression tests. Installation and configuration are explained clearly on the following site: http://webdriver.io/guide/services/visual-regression.html.

Using the following commands would help in comparing screenshots:

```
browser.checkViewport();
```

CHAPTER 13

Concurrency Testing Using Taurus and Selenium

Taurus is one of the best tools to run parallel or concurrent tests of selenium automated tests. Meaning, it can be used to run performance tests using automated test scripts (which are originally written for automated testing).

While running 'concurrent tests,' it is possible to find any defects such as misalignment of sections, object issues, disabled text boxes or hidden sections which are normally not possible to discover during functional testing.

Install Taurus from following website: http://gettaurus.org/install/Installation/.

Sample yml file of selenium to run single test on Taurus:

```
execution:
- executor: selenium
scenario: open_page

scenarios:
open_page:
browser: Chrome
timeout: 10s
think-time: 3s
requests:
- url: http://google.com
```

Sample yml file of selenium to run ten users' concurrent tests on Taurus 'simpleSeleniumTest.yml':

```
execution:
- executor: selenium
scenario: open_page
concurrency: 10
ramp-up: 0m
hold-for: 2m

scenarios:
open_page:
browser: Chrome
timeout: 10s
think-time: 3s
default-address: http://blazedemo.com
requests:
- url: /purchase.php
actions:
- waitByCSS(h2): visible
- keysByID(inputName): MyName
- clickByCSS(.btn.btn-primary)
assert:
- contains:
- 'Thank you for your purchase today!'
```

Command to run Taurus tests on command prompt:

```
bzt simpleSeleniumTest.yml
```

Results from Taurus test execution:

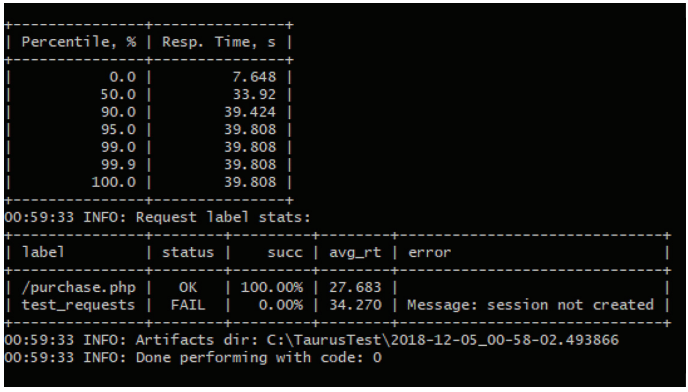

Figure 13.1. Test results.

CHAPTER 14

FREQUENCY OF ACCESSIBILITY TESTING

The most important question on non-functional testing (known as NFT) is, how frequent those NFT tests are to be executed during the lifetime of the web application? Once web applications are designed as per accessibility standards, those changes are going to be there in the application for rest of the application's life hence do they need to tested frequently to check if those standards are met or not?

Answer:

During 'User Story' Testing	During 'Functional' Testing	During 'Regression' Testing
Accessibility testing is required to prove early accessibility standard verification	Accessibility testing has to be performed manually using screen reading software such as JAWS, NVDA, and so on	Automate the accessibility tests by capturing html properties (related to accessibility tests) using selenium webdriverio in JavaScript and run these automated regression tests part of 'regression testing' cycle

14.1 TEST PLANNING ON ACCESSIBILITY TESTING-FREQUENTLY ASKED QUESTIONS

There is a small change request in my web application to replace an existing image with new image with background color change. Does this require a fresh cycle of accessibility testing?

Subset of accessibility tests related to color contrast analyzer need to be included part of test planning.

Web application has gone through an interface change and no code affected from front end layer—Does this require an accessibility testing cycle?
No (since GUI is not impacted part of the change)

Web application has been revamped with latest HTML5 technology and entire pages are tested individually when developed part of sprint that includes accessibility code scan using axe tool. Does this require an accessibility testing cycle?
Yes. Axe based code scan is basically html scanner to find any issues on overall properties. But accessibility behavior has to be tested part of exclusive manual and/or automated test execution cycle.

CHAPTER 15

Defect Classification

Every defect raised during test execution need clear inputs for stakeholders to understand and evaluate them according to project needs. If there are accessibility related defects it is highly recommended to follow some of the standards within defect fields mentioned as follows:

Defect Type: A or AA or AAA

Priority: High/Medium (for A and AA types of defects; mostly priority decided based on stakeholder's inputs to select either high or medium), Low (for AAA types of defects)

Severity: Severity 2* (for A and AA types of defects); Severity 3 (for AAA types of defects)

**Classification*: 1: Extremely Critical; 5: Least Impact

Accessibility Standard: It is important to mention in the defect on what accessibility standards (WCAG/Secton508) has not been met by the web application and respective references of the standards impacted.

Acceptance Criteria: Impacted acceptance criteria of the user stories are to be updated in this section. This would help to make the related user stories to be updated as BLOCKED or move to respective status. This is also another reason why the user stories should be updated with accessibility standards related acceptance criteria during user story design. This would help in testing accessibility early in the life cycle.

CHAPTER 16

Accessibility Certifications

Two types of certifications available for professionals to prove their proficiency in accessibility from International Association of Accessibility Professionals and preparation instructions can be referred at:

https://accessibilityassociation.org/certification

Websites adhere to accessibility standards can apply to WebAIM in order to receive site certification and more details are available at: https://webaim.org/services/certification/

CHAPTER 17

Advanced Selenium JavaScript Framework with Accessibility Tests

Writing JavaScript based selenium tests are fun and easy to build tests and run using packages such as cucumber-js. Prior knowledge of NodeJS development is required and suggested before getting on to the code examples given as follows:

Feature file:

Feature: Searching for GitHub projects
As an internet user
In order to find out more about GitHub projects
I want to be able to search for information about webdriverio

Scenario: GitHubwebdriverio search
When I search GitHub for "accessibility"
Then I should see some results

Scenario: 1.3.2 WCAG sequence test-Tab from username to email
When I click on username
Then I press tab key to navigate to verify placeholder of email as you@example.com

Scenario: GitHubwebdriverio search page Axe scan
When I search GitHub for "accessibility"
Then I should see some results
Then I should verify accessibility standards on the page

Step Definitions:

```
'use strict';
var{defineSupportCode} = require('cucumber');
var{By, until, Key} = require('selenium-webdriver');
var{expect} = require('chai');
varAxeBuilder = require('axe-core');

defineSupportCode(function({When, Then}) {

When(/^I search GitHub for "([^"]*)"$/, function (searchQuery, next)
{
this.driver.get('https://github.com/');
this.driver.findElement(By.name('q'))
.sendKeys(searchQuery);
this.driver.findElement(By.name('q'))
.sendKeys(Key.ENTER)
.then(function() {
next();
});
});

When(/^I click on username$/, function (next) {
this.driver.get('https://github.com/');
this.driver.findElement(By.id('user[login]' ))
.click()
.then(function() {
next();
});
});

Then(/^I should see some results$/, function (next) {
this.driver.wait(until.elementLocated(By.id('js-pjax-container')));
this.driver.findElements(By.id('js-pjax-container'))
.then(function(elements) {
expect(elements.length).to.not.equal(0);
next();
});
});

Then(/^I press tab key to navigate to verify (.*) of email as (.*)/, func-
tion(attribute,expRes,next){
```

```
varelement_to='user[email]';
varelement_from='user[login]';
this.driver.findElement(By.id(element_from))
.sendKeys(Key.TAB);
this.driver.findElement(By.id(element_to)).getAttribute(attribute)
.then(function (readValue) {
expect(readValue).to.equal(expRes);
next();
});
});

Then(/^I should verify accessibility standards on the page$/, function
(next) {
this.driver.then(()=>{
AxeBuilder(driver)
.analyze(results=>{
console.log(results);
})})
next();
});


});
```

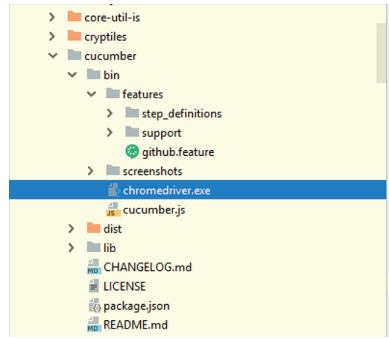

Figure 17.1. Chromedriver.

Steps to build this small group of tests:

- Install Nodejs on the target machine from https://nodejs.org/en/download/
- Install IDE such as WebstormIDE to write tests on JavaScript: https://jetbrains.com/webstorm/download/#section=windows

- Download and place the latest chrome driver for selenium at bin level as per the folder structure from http://chromedriver.chromium.org/downloads
- Update package.json of the project with tools such as this:

```
{
“name”: “seleniumtests”,
“version”: “1.0.0”,
“description”: “automated accessibility tests using selenium”,
“main”: “index.js”,
“scripts”: {
“test”: “node_modules/cucumber/bin/cucumber.js”
},
“author”: “Narayanan Palani”,
“license”: “MIT”,
“devDependencies”: {
“axe-core”: “^2.6.1”,
“chai”: “^4.1.2”,
“cucumber”: “^2.3.1”,
“pa11y”: “^4.13.2”,
“pa11y-reporter-1.0-json”: “^1.0.2”,
“sanitize-filename”: “^1.6.1”,
“selenium-webdriver”: “^3.5.0”
}
}
```

If tester write group of feature-based tests in cucumber and associate each line of feature file with respective step definitions (given earlier) and run the tests using commands such as from terminal, it is possible to execute the tests.

“C:\Program Files\NodeJS\node.exe” F:\SeleniumTests\node_modules\cucumber\bin\cucumber.js

Once the test is executed, please verify the test results such as,

```
3 scenarios (3 passed)
7 steps (7 passed)
0m21.430s
```

Screenshots of test execution are placed under ./cucumber/bin/screenshots/ folder.

17.1 CONSTRUCT ACCESSIBILITY TESTS WITH REUSABLE FUNCTIONS FROM CUCUMBER BOILERPLATE

Cucumber Boilerplate is a GitHub project with 150+ step definitions and associated functions written and used by engineers across the globe for webdriverio based tests. Using these step definitions will reduce test design and test execution time hence pre-written steps help in achieving accessibility verifications earlier rather than building the entire bunch of frameworks from the beginning!

Clone the following project and place the step definition files and support files from this repository to target repository or start writing tests within boilerplate folder structure:
https://github.com/webdriverio/cucumber-boilerplate

As instructed from GitHub cucumber-boilerplate framework, remove following files:
.travis.yml,
jest.json& wdio.BUILD.conf.js

Remove sample demo features from,
/src/features folder directory

Perform following two node installations to get the framework ready right away!
npm install yarn
yarn install

Test: Tab from link to UserID text box

Users with disability primarily concentrate on using keyboards and tab events to get to the place where they want to move on! Especially, bunch of Tab events will get them to reach user id text box in github.com page. There is a business link after several tabs and the next tab reaches user id field hence that is our next test to make sure pressing TAB from business link should get the user to user id field!

Feature file:

Feature: Test accessibility in GitHub login page
As a developer in test
I want to be able to test the accessibility of elements in login section of GitHub page

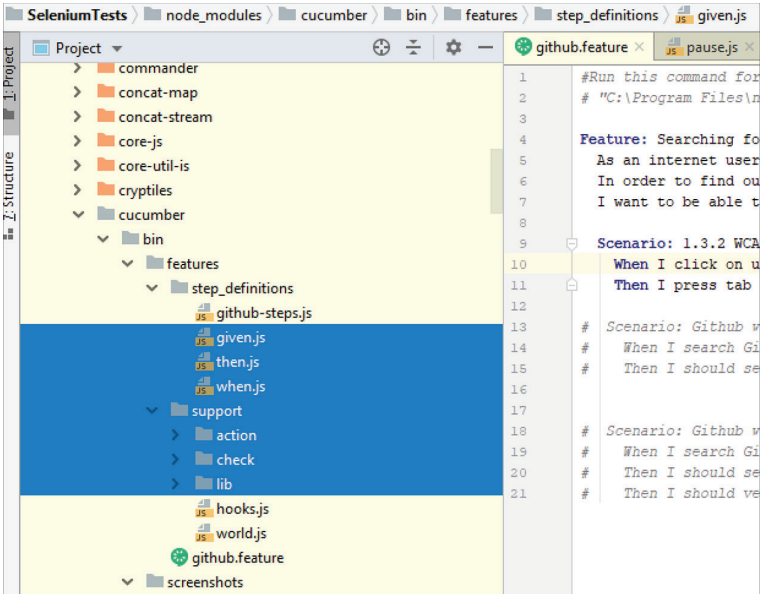

Figure 17.2. Test files.

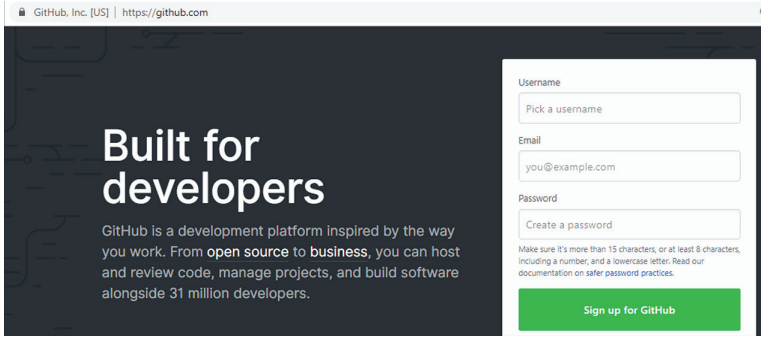

Figure 17.3. Github website.

Background:
Given I open the URL "https://github.com/"
And I pause for 1000ms

Scenario: Press tab from business link to reach user id text box
Then I scroll to the element "/html/body/div[4]/div[1]/div/div/div[1]/p/a[2]"
Then I press "Tab"
Then I expect that element "#user\5b login\5d" is focused

Note: Follow the page for further keyboard actions such as KeyA for letter A, and so on: https://w3c.github.io/webdriver/#keyboard-actions

Run this test using the following command in terminal of webstormIDE:

```
yarn run wdio --spec ./src/features/github_accessibilityTests.feature
```

Build a test to make sure the text box is not empty after entering texts in user id field:

```
Feature: Test accessibility in GitHub login page
As a developer in test
I want to be able to test the accessibility of elements in login section of GitHub page

Background:
Given I open the URL "https://github.com/"
And I pause for 1000ms

Scenario: Press tab from business link to reach user id text box
Then I scroll to the element "/html/body/div[4]/div[1]/div/div/div[1]/p/a[2]"
Then I press "Tab"
Then I expect that element "#user\5b login\5d" is focused

Scenario: Enter user id and move on to email field by pressing tab
Then I scroll to the element "#user\5b login\5d"
Then I press "KeyA"
Then I press "Tab"
Then I expect that element "#user\5b email\5d" is focused
Then I expect that element "#user\5b login\5d" is not empty
Then I expect that element "#user\5b email\5d" is empty
```

Similar Examples on www.google.com using various keyboard shortcuts:

```
Feature: Test accessibility in google page
As a developer in test
I want to be able to test the accessibility of elements in login section of google page using keyboard shortcuts
```

Background:
Given I open the URL “https://google.com/”
And I pause for 1000ms

Scenario: Press tab from business link to reach user id text box using tab key
Then I scroll to the element “//*[@title=”Search”]”
Then I press “KeyN”

Scenario: Read document title on google
Then I press “Insert, KeyT”
And I pause for 90000ms

Scenario: Press Insert and F7 to open Listed Links of NVDA
When I scroll to element “//*[@aria-label=”Homepage”]”
And I press “Insert,F7”
And I pause for 90000ms

CHAPTER 18

ACCESSIBILITY TESTING DURING "DESIGN" STAGE

Designing for yourself or your clients-accessibility is mandatory! All three elements should be focused to comply with accessibility standards:

- Design
- Development
- Content

Testing applications to serve users with vestibular disorder.

It is highly recommended to test applications and verify animations with following checklists: Allowing users to turn animation or flash content off?

- Application doesn't rely fully on animation?
- Application built with logical start and end states?
- Avoid using motion and tried using dissolve transitions?

If answered YES for all the aforementioned questions, then the test is pass for accessibility tests during design stage.

18.1 DESIGNING THE APPLICATION TO SERVE CUSTOMERS WITH 'COLOR BLIND'

Assume that the website selling bed sheet in three different colors Red, White, and Blue; While displaying the colors on the product page, bed sheet image has been displayed followed by three colors in squares with radio buttons.

What is the potential defect?

If multiple colors are provided on the page, it will be easy to choose the right color for normal user but all colors look same for users with color blindness.

Recommendation: Colors should be associated with texts next to it about the respective color names to let users with disability to listen and understand color differences.

18.2 DESIGNING AND TESTING APPLICATIONS WITH ANIMATION AND MOTION SENSITIVE

User animations responsibly to avoid accessibility issues and make sure the application doesn't rely on animations completely; Pattern-sensitive epilepsy-based users find it difficult when accessing when applications using flash or animation part of user journey!

18.2.1 TESTING FOCUS INDICATORS ON WEB PAGES

When users having diseases such as Arthritis, they don't use mouse most of the times and rely on using keyboards in accessing web applications. Also, cursor movement's changes when every keyboard action has been changed by keyboard press event. So, indicating where the focus on the web page is very important and very useful if the user has partial visibility.

Better code design to improve keyboard actions on web pages: Assume that the sections on the web page are provided with following html properties:

```
<div onclick="clickIt();">
```

While users are trying to access the section using mouse, it is possible to click on the web components but it is not possible to select the items when using keyboard. So it is highly advisable to get the code altered to,

```
<div href="#link" onclick="clickIt();">
```

18.2.2 HOW TO AVOID KEYBOARD TRAPS?

Assuming that customer filling a registration form on the web page and keeping the tax number as empty when this field is a mandatory field and there is an error message stating that the user should enter tax number

before navigating to other sections of the page. When user try to press 'TAB' or 'SHIFT+TAB' using keyboard, it is not letting user to navigate away from this text box. If user is using mouse, it is possible to click on other text boxes by mouse click event but not possible to move anywhere using keyboard until entering tax number on the text box. Issue: User is completely trapped in this situation and the defect is known as 'keyboard trap.'

Fix: It is highly recommended to define tabbing patterns by designers and get it shared to developers and testers hence keyboard traps can be easily identified and prevented in the code.

18.2.3 TOUCH-FRIENDLY INTERFACES

Assuming when user goes idle for some time, page displays 'Tap anywhere to continue' is displayed on a web page while accessing from laptop and user clicks on any location of the web page and the page gets activated again. If the same page is accessed by users using assistive technologies such as JAWS or NVDA, how will a user can click or Tap anywhere on the page to continue?

Solution: Instead 'Tap anywhere to continue' it should be a button (such as 'Click Here' or 'Continue') with link associated hence user can choose this by Listed Links of JAWS or NVDA.

18.2.4 VISUAL REPRESENTATIONS OF DATA

If there is a section of reports such as charts as interactive elements to show statistics and graphs, it will be complex for users with vision impairment to understand the meanings of the reports.

Solution: Table based report should be provided as an alternative to the Chart based reports hence, users with vision impairment can choose table to read and understand the reports.

18.2.5 FORM FIELDS LABELING

When a form provided with user id, password text boxes and login button, it is always recommended to provide label for user id and password text boxes which would help users with disability to read the labels through assistive technologies such as NVDA and JAWS to understand the purpose of it.

18.2.6 PLACEHOLDER

If the application has form fields to fill in, placeholder suggests what format the text or details are to be entered. For an example. Data of Birth field displaying DDMMYYYY and while user click on the field the placeholder disappears. It is different from label and disappears when the field has been focused. It is usually in lighter color and will be difficult for low vision user to read it.

Recommendation:
Make it readable for low vision reader with proper text color and rather than getting it disappearing while clicking or focusing on the field, try keeping the placeholder below the field when focused; So the placeholder won't disappear when focus is moved to respective field and user with vision impairment can understand better with placeholder to enter right details on the text box.

CHAPTER 19

ACCESSIBILITY CODE SCAN USING AXE PART OF AUTOMATED TESTS

Scanning a page using axe plugin is really good when there are static pages testing required. But most of the web testing scope needed solutions to test dynamic pages such as verify page three after successful login! If there are 800+ automated tests in which 150+ dynamic pages are to be scanned, it is not feasible to scan the pages and review them manually within the sprint schedule! Hence it is highly recommended to use axe part of automated test pack by implementing such as in the following.

19.1 JAVASCRIPT PROGRAMMING BASED AXE SCAN EXAMPLE

Pre-requisites:
After setting up the basic NodeJS based framework, please install selenium and axe using the following steps:
Install Selenium using the following command:

```
npm install selenium-webdriver --no-save
```

Install Axe using the following command:

```
npm install axe-webdriverjs
```

Sample Code:

```
varAxeRunner = require(‘axe-webdriverjs’);
varWebDriver = require(‘selenium-webdriver’);
```

```
varaxeFileSource = require('fs').readFileSync('./node_modules/axe-
core/axe.js', 'utf8')

browser = new WebDriver.Builder()
.forBrowser('chrome')
.build();

browser.get('https://github.com/')
.then(() => {
// Set axe builder on top of chrome driver to scan:
const axe = new AxeRunner(browser, axeFileSource)
// Run axe tests on the chrome driver to provide results
axe.analyze(function (results) {
console.log(results);
});
});
```

19.2 JAVA PROGRAMMING BASED AXE SCAN EXAMPLE

Pre-requisites:
axe.min.js is a minified version of original axe scanning script and it is available part of API packages from https://github.com/dequelabs/axe-selenium-java/tree/develop/src/test/resources

Sample code:

```
packageAccessibilityPageScan;
importorg.testng.Assert;
import java.net.URL;
import static org.junit.Assert.assertTrue;
importcom.deque.axe.AXE;
importorg.junit.rules.TestName;
importorg.junit.Rule;
importorg.junit.Before;
importorg.junit.After;
importorg.junit.Test;
importorg.json.JSONArray;
importorg.json.JSONObject;
importorg.openqa.selenium.WebDriver;
```

```
importorg.openqa.selenium.chrome.ChromeDriver;

//Section1:
public class codeScanAccessibility {
//Section2:
      @Rule
      publicTestNameautotestName = new TestName();
//Section3:
      private static final URL axeCodeScanCode = Test_Accessibility.
class.getResource(“pathToDriver/axe.min.js”);
//Section4:
      @Test
      public void testAccessibility ()
      {
            WebDriver browser;
            System.setProperty(“webdriver.chrome.driver”,”path-
            ToDriver\chromedriver.exe”);
                  browser = new ChromeDriver();
                     browser.get(“https://github.com/”);
//Section 5:Axe based violations checker:
            JSONObjectresponseJSON = new AXE.Builder(browser,
            axeCodeScanCode).analyze();
//Section6:
            JSONArray violations = responseJSON.getJSON
            Array(“violations”);
            if (violations.length() == 0) {
                  assertTrue(“No accessibility issues”, true);
            } else {
                     AXE.writeResults(autotestName.getMethodName(),
                     responseJSON);
                     assertTrue(AXE.report(violations), false);
            }
      }
}
```

Code explanation for each section:

1. This is the class file in which axe scan functions are initiated using axe scripts provided from git-hub repositories.
2. Test is initiated in order to write the accessibility results later after scan completion.

3. After taking axe.min.js from GitHub projects such as axe-selenium-java, place it in the project path and update the section with right location hence test can pick the file part of axeCodeScanCode.
4. Regular browser initiation part of selenium with path of chrome browser driver. If Internet Explorer or Firefox or any other browser needs to be used, kindly update this section with right browser driver details from your automated test repository.
5. This part of the code scans the page for any accessibility violations.
6. Axe based accessibility reports are generated part of this script. Not every violation turns out to be a defect. This report has to be reviewed along with developers to understand how many are to be considered to review further for code fix.

CHAPTER 20

ACCESSIBILITY TESTING ON ERROR MESSAGES

This is one of the important sections of accessibility testing on verifying error messages and proving that users with disability can understand on what exactly gone wrong?

Following examples provide clear description of important test scenarios to be covered part of error messages testing.

20.1 PLACEMENT OF THE ERROR MESSAGES

Assuming that the web application contains list of fields and user entered incorrect text as 'a$b' on 'city of birth' field hence there is an error message displayed on top of the web page on bold red color font with the following words and focus moves to error message after pressing 'Tab' from 'City of Birth' text box:

20.1.1 "ERROR: PLEASE ENTER RIGHT TEXT ON THE FIELD"

What is the problem in this error message with respect to accessibility?
Issue1: Error message is not explaining about particular text field 'city of birth'
Issue2: Error message is not placed near to 'City of Birth' text field hence user has to find a way to navigate back to this text field
Issue3: Focus goes to error message hence user lost the navigation flow from field to top of the page and user has to navigate to field after several steps to reach 'City of Birth' to enter right text.

Location of error messages, current focus and respective fields to remove errors.

Assuming that the user story or requirements says to verify right error message; When user enter incorrect text in 'City of Birth' field and press 'Tab' that leads to display error message next to field at right hand side but the focus lost rather than going back to 'City of Birth' field.

What is the problem in this error message with respect to accessibility?

Issue1: Focus is lost rather than getting the focus to next form field and bringing the focus back to 'City of Birth' field.

Issue2: Error message has been displayed as per user story or requirement but the focus is not discussed in detail in the requirement hence it is not a valid defect to raise against this requirement.

Recommendation: Designer and developer related to this user story has to be involved while discussing the 'Issue2' focus issue and raise this focus issue as an 'Issue' or 'Accessibility Recommendation' and get the user story updated on 'Acceptance Criteria' to validate the focus flow when 'Tab' is entered;

Recommendation verification points while testing error messages

- Verify that the error icon is highly recommended to be placed next to error message or the relevant field.
- Language of error messages should be plain and simple and readable for users with disability to understand to fix the error in right field.
- Color of error messages should be consistent and clearly visible to users with right font size.
- After error messages displayed on the page, please make sure the focus is not lost and user is navigated to appropriate locations as expected; If this is not detailed in user story, please discuss with teams to get the story updated with right focus instructions.

Conclusion

Accessibility testing is one of the emerging testing practices and it is expected to be a niche software engineering job in the next ten years' time since approximately 1 in 5 users generally need some sort of accessibility while using computer and handheld devices.

Though it is possible to automate selected accessibility tests such as keyboard navigations, it is advised to automated after few runs of manual test execution to observe and understand user experience on the web application with the help of accessibility test cases.

"No future for manual testing" is a clear myth! Accessibility is expected to be the next generation manual testing and we all have to be proud of such knowledge and experience to perform accessibility testing and prevent defects to provide (disabled) users a 'best in class' experience.

Accessibility related defects lead to pain and hard times for users with disabilities; Defects such as poor color contrast will create greater pain in disabled users with low vision and lead to further complications on their eyesight; It is not only our duty to perform accessibility testing, but also it is our primary responsibility to prevent accessibility defects from any electronic devices which will help users with disability to undergo good experience!

Creating accessibility testing related business analysis, design, development and testing jobs are 'extremely' important for leading organizations and it is 'highly' recommended to recruit users with disabilities to undergo accessibility testing training and test the web application to explore defects which would be a greater advantage to test early and release with no accessibility defects! Training functional testing professionals on accessibility would help them to master in the web accessibility, perform accessibility test execution and reduce the defects on every product release!

As a leading software testing industry expert, I have come across world leading 'accessibility testers' and their sessions in the recent days.

After being trained by, Chandrashekar Korlahalli (Mr. CBK) and Yamini Gupta on accessibility testing, I have been experiencing test case design, defect retests, defect prioritization, code analyses related to accessibility tests. Manoj Kumar (from applitools) has inspired thousands of audiences through his web live session on 'Automated Accessibility Testing' during the year 2017 and that has led me to experiment automated tests on accessibility. After completing few manual test executions of accessibility test cases, I could write automated tests using tools such as webdriverio, cucumber and gulp and run those automated tests through Continuous Integration/Continuous Delivery (CI/CD) solutions such as Jenkins pipeline. My sincere thanks to audience for reading this book. My humble request is to start implementing accessibility testing in the testing assignments to prevent any further accessibility defects if this is not planned already. My vision is to implement accessibility tests across customer facing software projects worldwide through my public awareness sessions and this particular book release hence it would help creating a fair world for customers with various disabilities!

This book can be gifted to 'computer science' or 'software engineering' students to learn and find jobs in I.T industry!

This book can be a gifted to charities which are helping people with vision impairment and other disabilities hence they can distribute it to people with disabilities to learn and earn a work opportunity in accessibility testing! If organizations recruit eligible candidates with disabilities, it will be of great help for them to test and spot accessibility defects on time! In order to create employability among users having disabilities, it would a great idea to organize 'accessibility testing' training sessions from charities and non-profit organizations for the registered users hence organizations can contact charities to get connected to trained users for recruitment.

Hope you enjoyed reading the book and please don't forget to provide feedback for this book with detailed review comments on online retailer websites such as Amazon.

I pledge to create a better world for users with disabilities through this book-hoping to see your help in extending your technical arms to those special users via strong accessibility testing coverage!

Abbreviations

AC	Acceptance Criteria
AUT	Application Under Test
CI/CD	Continuous Integration/Continuous Delivery
DoD	Definition of Done
FF	Firefox
HTML	Hyper Text Mark-up Language
IE	Internet Explorer
IT	Information Technology
NVDA	Non-Visual Desktop Access
QA	Quality Assurance
UFT	Unified Functional Testing
URL	Uniform Resource Locator
WCAG	Web Content Accessibility Guidelines
WAT	Web Accessibility Toolbar

About the Author

As a proven software testing industry leader, **Narayanan Palani** volunteered to share technical knowledge through his best-selling book series 'Software Automation Testing Secrets Revealed' and he has been endorsed by Tech-Nation as Exceptional Talent of UK within Digital Technology; His well-designed Github open source testing projects are accessed worldwide and helping job seekers to get entry into junior testing jobs.

Being a leading test specialist holding eleven awards, seventeen international certifications, eight published research papers and five books selling across the world, he continued to drive teams towards cutting edge technology implementations as a Chapter Lead for Quality Engineering!

INDEX

THIS TITLE IS FROM OUR COMPUTER ENGINEERING FOUNDATIONS, CURRENTS, AND TRAJECTORIES COLLECTION

Mohammad Noori, *Editor*

Momentum Press is one of the leading book publishers in the field of engineering, mathematics, health, and applied sciences. Momentum Press offers over 30 collections, including Aerospace, Biomedical, Civil, Environmental, Nanomaterials, Geotechnical, and many others.

Momentum Press is actively seeking collection editors as well as authors. For more information about becoming an MP author or collection editor, please visit http://www.momentumpress.net/contact

---

Advanced Selenium Web Accessibility Testing

Software Automation Testing Secrets Revealed

Narayanan Palani

If you are searching a topic on Google or buying a product online, web accessibility is a basic need. If a web page is easier to access when using a mouse and complex to navigate with keyboard, this is extremely difficult for users with disabilities. Web Accessibility Testing is a most important testing practice for customers facing web applications.

This book explains the steps necessary to write manual accessibility tests and convert them into automated selenium-based accessibility tests to run part of regression test packs. WCAG and Section 508 guidelines are considered across the book while explaining the test design steps.

Software testers with accessibility testing knowledge are in high demand at large organizations since the need to do manual and automated accessibility testing is growing rapidly. This book illustrates the types of accessibility testing with test cases and code examples.

As a proven software testing industry leader, **Narayanan Palani** volunteered to share technical knowledge through his best-selling book series, *Software Automation Testing Secrets Revealed*. He has been endorsed by Tech-Nation as exceptional talent of UK within digital technology. His well-designed Github open source testing projects are accessed worldwide and helping job seekers to get entry into junior testing jobs.

Palani is a test specialist who holds eleven awards, seventeen international certifications, eight published research papers, and five books selling across the world. He continues to drive teams towards cutting edge technology implementations as a chapter lead for quality engineering.

MOMENTUM PRESS
ENGINEERING

ISBN: 978-1-94944-943-3
90000
9 781949 449433