Automated CSS Testing by HTML Tags for Verification

Dheeraj Tatipamula  
 Computer Science  
 University of Texas at Arlington  
 Arlington Texas USA  
 dxt3485@mavs.uta.edu

Harini Muddana  
Computer Science  
 University of Texas at Arlington  
 Arlington Texas USA hxm7426@mavs.uta.edu

ABSTRACT

As a part of UI development unit testing is an important task to check which things are visually missing according to design or website consistency.

As a part of this paper, we want to develop an algorithm which will take the content of website and test them with given values by user and generate report of anomaly. Then user can check and verify the report given.

We implemented this algorithm which goes through all CSS and checks with pre-defined standards given by the user. Regardless of where in the DOM the HTML tag is present.

Initially we will try developing the tool to find color and font anomalies in the webpage. Algorithm we develop will take CSS of the webpage and process in algorithm.

We added Red Background color to tags which show difference for font-sizes. We added green background for line height difference. We added green background for line height difference.

CCS CONCEPTS

software and its engineering →Graphical UI languages; Software maintenance tools

KEYWORDS

accessibility, usability, verification, SMT, layout, CSS, semantics, HTML DOM, DOM manipulation, JavaScript, jQuery.

ACM Reference format:

Dheeraj Tatipamula, Harini Muddana. 2018. Insert Your Title Here: Insert Subtitle Here. In *Proceedings of ACM Woodstock conference (WOODSTOCK’18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1  Introduction

 It is critical to ensure that a website's appearance is accurate for it to be successful. According to a recent study, a user often forms an opinion about a website within the first 50 milliseconds, based solely on the "look and feel" of the website.

Organizations spend a lot of time and resources on testing these failures, manually testing all these CSS errors is very difficult for the testing team as it contains lot of content and pages.

Sometimes the content is jumbled in between the sentences, and it becomes difficult for the person to test each line in the page. This might lead to presentation error.

Manually troubleshooting such problems is time-consuming and error-prone, and current methodologies do not offer an automated solution.

It is very difficult to find minute errors in CSS with respect to design for example, font size-12 and font size-14 in websites with huge content cannot be differentiated. It manually takes very long time for a person to go through all the content to find the differences.

We thought of 2 possibilities for this, by using image of the webpage or directly taking CSS and HTML file from browser to run through the algorithm. We are taking HTML and CSS of the page and executing our algorithm to find the discrepancy in the font sizes for pre-defined values.

1.1 Testing Input and Output

For this algorithm we must give an input which is html file with CSS included externally or internally for the website. And we give output as changes on the website as discrepancies are added with red, blue, and green background colors. We only have internally JavaScript so; user must add his values for default and general in place of default font sizes and add his html code and CSS code in the below html tags and CSS file.

*1.1.1 Input.* Here I will add images of input and what is the changes required to test their website using this algorithm.

Text

Description automatically generated

Figure 1: Image where we are adding default font sizes for the website and developer can add this and test.

A screenshot of a computer

Description automatically generated

Figure 2: Image where we are displaying the website.

*1.1.1 Output.* Output can be on both report format for the user or adding background colors to the DOM as we consider for the project. We cannot add CSS if we want to do it for external website and we consider for this project only using internal JavaScript for testing its UI.

Background pattern

Description automatically generated

Figure 3: Image where we show to use what all the text are not in sync with default for that tag in website.

1.2 Case Study.

In this paper we consider one thing that. It tests for only font size and font-family for the text. We are not adding all major CSS test cases and checking variation.

As we know visual UI testing is very tough using automation and it takes lot of scenarios to test the UI without manual interruptions is very generic for all the test cases.

1. Take the input from user.
2. User gives all default values to the website pages and test it according to default values.
3. Get the tags and text values where it is not matching with default values.
4. Add the background color for all those tags in DOM.

We will get DOM access to the HTML and loop through all CSS elements of the website and check for the discrepancies.

2  Background and Related Work

WEBSEE:[1]

WebSee's method focuses on recognizing and localizing display faults in web pages automatically. It compares the visual representation of the delivered web page under test with its appearance image to discover presentation faults using computer vision techniques. The technique then creates rendering maps of the test web page to identify the HTML components that are responsible for the presentation faults in the specified locations of the web page.

This method requires three inputs: (1) a test web page in the form of a URL pointing to a network or filesystem location, (2) an appearance oracle in the form of an image that can be either a mockup or screenshot of a previously correct version of the test web page, and (3) a set of special regions that define dynamic areas of the test web page

Such as advertisements and news feeds are popular features in current web applications. Our method works through three phases:

(1) detection (comparing test web page rendering to oracle),

(2) localization (identifying potentially faulty HTML elements responsible for the observed presentation failures in phase 1), and

(3) result set processing (prioritizing the set of potentially faulty HTML elements identified in phase.

The user receives the output of phase 3.

Verifying That Web Pages Have Accessible Layout[2]

Usability and accessibility rules strive to make graphical user interfaces accessible to all users by demanding, for example, that text be large enough to read, interactive controls be visible, and heading size match to relevance. These rules must apply to the unlimited number of possible web page renderings caused by different screen sizes, typefaces, and other user choices. Because 1) the recommendations are not written in a formal language,

2) the semantics of browser rendering are not well known, and

3) no tools exist to evaluate all possible renderings of a web page; these principles are currently tested by manual inspection of a few renderings. These issues are addressed by VizAssert. It begins by introducing visual logic, which allows users to exactly specify accessibility features.

Diagram

Description automatically generated

Figure 4: VizAssert ensures that web pages satisfy accessibility guidelines. VizAssert transforms those assertions, expressed in visual logic, into properties of the HTML and CSS source of the web page, then encodes those properties to formulas in the quantifier-free theory of linear real arithmetic.[2] referred this image from there paper.

**HTML**

HTML defines elements and text. Each element has a tag name, and text is placed within and between elements. For example, the following HTML represents 4 elements (with tags html, body, b, and button) and four pieces of text:

<html><body>This is <b>formatted</b> text

and a <button>button</button></body></html>

Some HTML elements, like the button element, are specially interpreted by the browser and rendered with browser- or OS-specific methods. Most other elements have no special behavior: browsers provide a default CSS file to ensure that, for example, text inside a b-tagged element is bold. It is this default CSS file, not something intrinsic to the b tag, that causes the text to render in bold.

**CSS**

A CSS file defines a sequence of rules. Each rule selects certain elements and then sets the values of various properties on those elements. The following CSS file contains two rules:

body {margin: .5em;}

b {font-weight: bold;}

The first rule selects elements with the body tag and sets their margin property. The second rule selects elements with the b tag and sets their font-weight property. CSS also allows selecting elements by their relationship to other elements or by attributes attached to those elements. If two applicable

3  Algorithm and Working Implementation

Algorithm:

The first step in this algorithm is to take HTML and CSS from the website pages and add JavaScript code to this, we can add additional test cases also by giving default sizes and default values to test the webpage.

The major technical challenge we faced during this project implementation is, it took lot of time to get the DOM tree from the website. And looping through all the content of the DOM to find the differences with the pre-defined values.

We implemented this algorithm which goes through all CSS and checks with pre-defined standards given by the user. Regardless of where in the DOM the HTML tag is present.

Implementation steps:

1. Get all the content from present HTML and CSS for this website
2. User will give default values for font sizes and font weights.
3. Go through all different tags and compare the tag CSS on the website for given default value. If we see any discrepancy in the website CSS, we add background color for that.
4. We add different background color for different type of mismatch we find. So that user will not what to change in the text CSS to and what paragraph it is overriding CSS is applied because of class hierarchy.

Working.

Text

Description automatically generated

Figure 5: sample code checking ‘p’ tags.

Here we can see in this code it considers P tags for the website. It loops through all the p tags on the website and checking for discrepancy. Here we added just sample for 1 tag and 1 check scenario we can improve implementation of the algorithm with multiple checks and multiple scenarios.

We need to know where the font size is overridden and where we need to change it, for this reason we loop through all elements with specific tag regardless of what block they present on website once and check for this.

Text

Description automatically generated

Figure 6: The above paragraph is used font 14 instead of default font size given by user so we added red background.

When user corrects the CSS and reruns the website the background color is removed.

Graphical user interface, text, application

Description automatically generated

Figure 7: The above paragraph is used line height 1 rem instead normal given by user, so we added green background.

Graphical user interface, text, application

Description automatically generated

Figure 8: The above paragraph is used different font family instead normal given by user, so we added blue background.

The major aspect of this project finding this type of error in one click instead of going through whole website and check at each point or section for this minimal visual errors.

We can add more CSS conditions and check for different test cases in tags we implemented code and checked for headers and p tags and produced the output in implementation.

Similarly, our future work section says what we plan to do in the next module of implementation.

4  Technical challenges

The major technical challenge we faced during this project implementation is, it took lot of time to get the DOM tree from the website. And looping through all the content of the DOM to find the differences with the pre-defined value is also time taking.

As you can see in our solution, we added red background for the tags which are not in accord with the design, but we cannot do this for the external websites, we are planning to test by taking the inputs from the user.

5 Conclusion: In this project, we created our own webpage using HTML tags and CSS, and we implemented our own algorithm to run through all the tags of given webpage to extract the CSS of each tag and then compared the values from the CSS to the pre-defined values given by the user. Those values which are not in accord for their HTML tags are added with red and blue background colors for font size, font weight discrepancies, respectively.

By this, we can test any web page for design values within efficient time and less error prone compared to manual testing.

6 Future Work:

We like to implement many things for this in future

1. Our major challenge is to take external website CSS and HTML of an external website and test with user defined values for default values for tags and give the error list to user since we cannot add background colors to external website using line number.
2. We can add many more functionalities, since we tested only for some html tags such as header, paragraph and we only checked for some CSS elements such as fonts and line height we can extend the scope by adding many CSS elements
3. We are planning to add this to JavaScript plugin and where user can use the plugin and test according to before even publishing the page
4. Checking alignment with help of tools like Websee and extending the functionality to the responsiveness of the screen

6 TOOLs used:

Languages

 HTML

 CSS

 JavaScript

 jQuery

IDE

 Visual Studio Code

Console window to test our application of present changes.

REFERENCES

[1] WebSee: A Tool for Debugging HTML Presentation Failures. Sonal Mahajan and William G. J. Halfond University of Southern California

.

[2] Verifying That Web Pages Have Accessible Layout. Pavel Panchekha Adam T. Geller Michael D. Ernst Zachary Tatlock Shoaib Kamil

[3] C. Eaton and A. M. Memon. An Empirical Approach to Testing Web Applications Across Diverse Client Platform Configurations. International Journal on Web Engineering and Technology (IJWET), Special Issue on Empirical Studies in Web Engineering, 3(3):227–253, 2007.

[4] Apple. 2017. Human Interface Guidelines. https://developer.apple. com/ios/human-interface-guidelines/

[5] Apple. 2017. UI Design Do’s and Don’ts. https://developer.apple.com/ design/tips/

Conference Name:ACM Woodstock conference

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00