Bachelor's thesis

Information and Communications Technology

2019

Binh Bui

WEB COMPONENTS TESTING PROCESS

– integrating Web Components into Vaadin

TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

ICT

2019 | 55 pages, 5 in appendices

Binh Bui

WEB COMPONENTS TESTING PROCESS

- integrating Web Components into Vaadin

As the need for better presentation and interaction of web sites grows, the average codebase is growing larger and more sophisticated. Web Components standards were proposed in 2011 to enable creating re-usable components for saving developing and maintaining resources. Google adopted the standards rather early and used their products as a catalyst for other browsers and web frameworks to join in supporting Web Components.

Vaadin is a Java web framework, which got its reputation as a great alternative to JavaScript web frameworks. However, Vaadin needs to support the Web Components standards to stay relevant in the framework scene. As a result, Vaadin conducted quantitative research on testing hundreds of web components to collect more intelligence on the standards, as well as to gather a set of quality components to be imported into Vaadin Directory, a components marketplace for Vaadin.

The goal of this thesis was to give a general understanding of the Web Components standards along with Vaadin framework and Vaadin Directory. Then, this thesis aimed to find the usage share of Web Components libraries to determine which library Vaadin should analyze. These goals were achieved by testing hundreds of web components to collect each component's usage problems and gather general statistics.

The whole testing process was completed in 3 months with plenty of useful findings and analytics data obtained. A collection of screenshots of web components preview and a collection of web components online demos were also acquired. Lastly, Vaadin Directory became populated with a set of valuable components.

This thesis reached a domain that has never been touched before, which is to test hundreds of web components. As a result, the obtained data was beneficial for helping Vaadin integrate the Web Components standards and develop a comprehensive set of Vaadin web components.

KEYWORDS:

Web Components, Vaadin, Polymer, PWA, JavaScript, Testing

CONTENTS

APPENDICES

FIGURES

TABLES

LIST OF ABBREVIATIONS

Android	Android is a mobile operating system developed by Google.
Angular	Angular is an open-source Typescript framework for building mobile and desktop web applications.
API	Application Programming Interface is a set of functions and procedures, allowing the creation of applications that access the features or data of an operating system, an application, or other services.
App	App is short for Application.
Aurelia	Aurelia is a client-side JavaScript framework to create web applications.
Bower	Bower is a web package manager that handles all the external dependencies like frameworks, libraries, assets, utilities.
CDN	Content Delivery Network is a geographical system of distributed servers that help deliver pages and content to a user.
CodePen	CodePen is an online code editor and an online community for front-end web development to test and showcase user-created HTML, CSS, and JavaScript code snippets.
Chrome	Google Chrome is a web browser developed by Google.
CSS	Cascading Style Sheets is a language that delineates the presentation of an HTML document.
DOM	Document Object Model is a cross-platform interface to support HTML and XML documents.
Edge	Microsoft Edge is a web browser developed by Microsoft.

Ember	Ember is an open-source JavaScript framework for creating web applications.
ES6	ECMAScript 6, also known as ECMAScript 2015, is a scripting language specification that is standardized by ECMA International.
ES Modules	ECMAScript Modules is one of Web Components standards, which defines the inclusion and reuse of JavaScript documents in a modular way.
Firefox	Mozilla Firefox is a browser developed by Mozilla Foundation and Mozilla Corporation.
GitHub	GitHub is a version-control code hosting platform that helps developers store and manage their code, as well as collaborate and share code.
GWT	Google Web Toolkit is an open-source Java-based framework for creating AJAX web applications. It allows developers to maintain JavaScript front-end applications in Java.
HTML	HyperText Markup Language is a standard markup language used to create web pages. It describes the structure of the document of the web page.
IE11	Internet Explorer 11 is the final version of the Internet Explorer web browser developed by Microsoft.
Java	Java is an object-oriented programming language that is class-based, and designed to have the least amount of implementation dependencies. It is used to create software for multiple platforms.
Javadoc	Javadoc is a documenting tool for the Java language that parses the declarations and documentation comments in HTML format.

JavaScript	JavaScript is a dynamic computer programming language that conforms to the ECMAScript specification. It is often used to enhance web pages.
jQuery	jQuery is a JavaScript library designed to make it easier to manipulate HTML DOM tree and use JavaScript on the web.
JS	JS is short for JavaScript.
JSON	JavaScript Object Notation is a lightweight format for storing and transporting data. It is commonly used when the data is sent from a server to a web page.
Node.js	Node.js is an open-source JavaScript framework to help build network applications.
Opera	Opera is a browser developed by Opera Software.
pandas	pandas is an open-source Python library that provides data manipulation and analysis.
Polyfill	Polyfill is a piece of code, usually JavaScript, used to provide a feature on web browsers that do not natively support it.
Polymer	Polymer is an open-source JavaScript library for building web components.
Regex	Regular Expression a special text string that defines a search pattern.
Safari	Safari is a web browser developed by Apple.
SemVer	Semantic Versioning is a versioning system that defines the nature of change in a release directly into the version string.
simplejson	simplejson is a Python library for dealing with JSON. It can encode and decode JSON.
SkateJS	SkateJS is an open-source JavaScript library for building web components.

Slim.js	Slim.js is a JavaScript library for developing native web components.
Spec	Short for Specification.
StencilJS	StencilJS is a compiler to help build web components.
String	String is a data type used in programming. It usually represents a sequence of characters.
Tech	Tech is short for Technology.
UI	User Interface is the space where humans interact and communicate with the devices.
URL	A Uniform Resource Locator is a reference to a web resource on the Internet.
urllib	urllib is a Python core library for handling modules.
Vaadin	Vaadin is a Turku-based Tech company that offers an open-source framework with the same name as its main product, the Vaadin framework. The Vaadin framework is an open-source Java framework used to create web applications. The term Vaadin is used interchangeably between the company and the framework in the thesis.
Vaadin Directory	Vaadin Directory is a marketplace for Java and JavaScript user interface components.
VanillaJS	VanillaJS is plain JavaScript without any additional libraries or frameworks.
VirtualBox	VirtualBox is an open-source software for virtualizing x86 computing architecture. It helps to emulate an operating system.
Vue	Vue is an open-source JavaScript framework for building user interfaces and single-page applications.
Web	Web is short for Website.

Web Components	Web Components, with a capital ‘W’ and ‘C’, is a suite of web standards, which allows you to create reusable elements.
Web component	Web component, with lowercase letters, is a custom element that is produced by using Web Components standards.
WebKit	WebKit is an open-source browser engine developed by Apple. It is currently used by Safari and formerly used by Google Chrome.

1 INTRODUCTION

As the need for better presentation and interaction of web sites grows, the average codebase is growing larger and more sophisticated. Therefore, the idea of re-using code has been highly prioritized to save resources for maintaining. Web Components standards were proposed in 2011 to actualize that idea. Web Components are a suite of different standards that allow the creation of reusable custom elements (MDN 2019e).

From the moment Web Components were proposed, they have become more popular and widely adopted. By 2017, all modern browser vendors have either implemented the standards natively or planned to (Polymer 2018a, MicrosoftEdge 2015, Kesteren 2014). On the same note, many JavaScript frameworks have supported the integration of web components such as Angular, Vue, Ember, or Aurelia (Eisenberg 2019, Angular 2019, Schooley 2015, Gore 2018).

Being a competent web framework in the scene, Vaadin wanted to integrate Web Components standard into Vaadin framework. Besides, Vaadin planned to import a significant number of web components into Vaadin Directory once the integration completed. The primary purpose is to support its community with plenty of ready-to-use components and encourage more web developers to try out Vaadin.

As Vaadin wants to collect more intelligence on the standards, the author conducted quantitative research on web components. The research involved testing hundreds of components from the website webcomponents.org for two central purposes: to gain a satisfying level of knowledge on the standards and to gather a collection of valuable web components for importing into Vaadin Directory. This testing process happened in 2017.

The goal of this thesis is to give an overall understanding of the Web Components standards along with Vaadin framework and Vaadin Directory; then, demonstrate the web components testing process. Extra statistics extracted from the process are delineated at the end of the thesis.

Since Web Components are already a popular topic by that time and will most likely continue evolving, a publication on this topic does not stay relevant for long. Instead of diving deep into the standards, this thesis aims to share the knowledge and experience from the quantitative testing process.

There is one recent thesis published in 2019: “Web Components” (Bui, T. 2019), written by Tan Bui, which delved deeper into Web Components specifications and their usage. Also, the thesis features some application prototypes which apply web components, along with the instruction on how to deploy the application using the Heroku service.

This thesis is divided into 7 chapters. The beginning chapters, 1, 2, and 3 focus on giving a general overview of Web Components, Vaadin, Vaadin Directory, and their relations with each other. Chapter 4 aims to find the usage share of Web Components libraries. Chapter 5 describes the prerequisites for the testing process, while Chapter 6 dives deeper into the whole testing process. Chapter 7 presents the testing results, the obtained statistics, and the importing web components to Vaadin Directory. Finally, Chapter 8 concludes this thesis with some key points taken from the whole testing process.

2 FROM PROPOSED STANDARDS TO BEING NATIVELY ADOPTED

Web Components are a suite of web platform Application Programming Interfaces (APIs) that allow developers to encapsulate website codes into a single component. The APIs make it easier to reuse the component in web applications (MDN 2019).

Alex Russel first introduced Web Components at the Fronteers Conference in 2011 (Fronteers 2011). Later, the standards became so popular that modern browser vendors started to implement them natively.

The most significant factor for the standards' rise in popularity was when Google released Polymer in 2013, a tool that aims to simplify the creation of web components (Cimpanu 2015). This event led to several other libraries created to help developers create web components, such as StencilJS, Skate, and Slim.js.

As Google wanted to catalyze the adoptions of Web Components, the Polymer team also created a website, called webcomponents.org, to provide a place for developers to publish and share web components.

2.1 Web Components standards

The Web Components standards (Figure 1) consist of four key features:

1. Custom Elements – enabling the creation of new Hypertext Markup Language (HTML) element with custom names and behaviors (e.g., <my-elem>) (WebComponents 2018).
2. Shadow Document Outline Model (Shadow DOM) – encapsulating the markup and style to only inside the element (WebComponents 2018).
3. HTML templates – defining template in HTML that would only be rendered upon calling (WebComponents 2018).
4. HTML Imports – enabling reusing and including of HTML documents in other HTML documents (MDN 2019c).

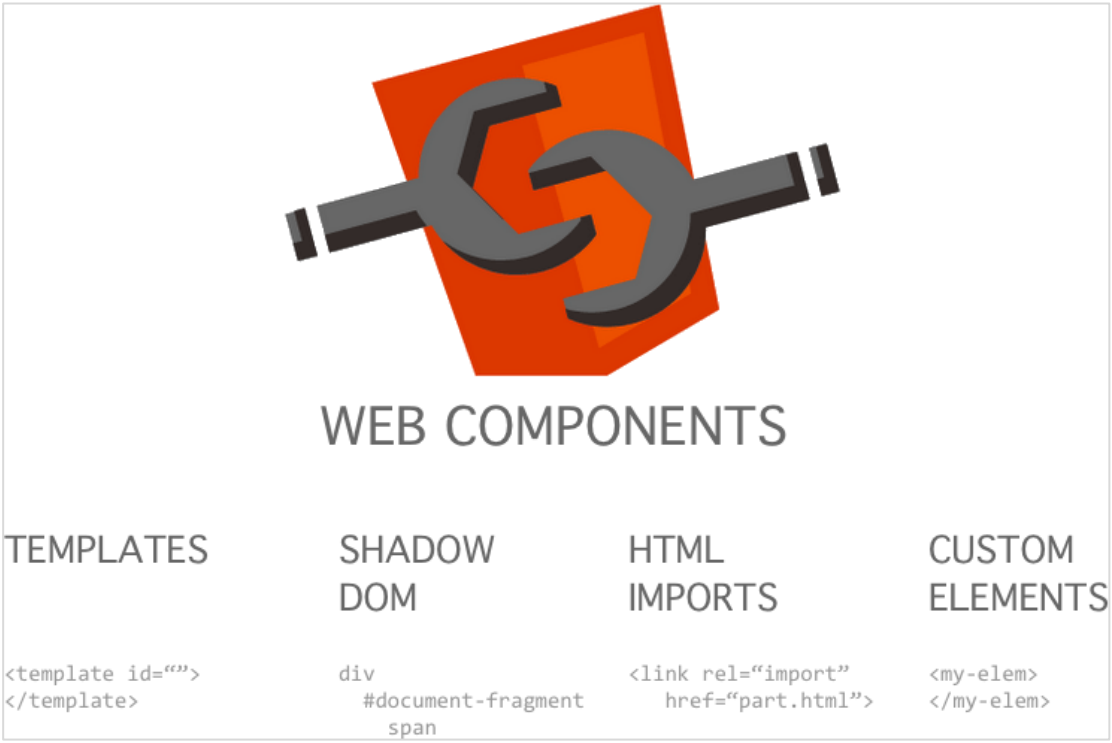


Figure 1. Web Components standards (Carl 2018).

Those mentioned standards were the first draft of the proposal, which still included HTML Imports. In the latest 2019 specifications, ECMAScript Modules (ES Modules) has replaced HTML Imports (Chrome 2017). Since the testing process happened in 2017, all the web components were using HTML Imports.

2.2 Popularity of Web Components

The idea of writing reusable and portable components for the web has been around for a long time. Famous JavaScript frameworks, such as Angular and React were adopting the idea rather early. However, there were no standards to achieve interoperability between different frameworks. The whole adoption was moving at a slow pace. With Web Components standards becoming accepted, the development and usage of creating reusable components have become more transparent and straightforward (Revill 2016).

2.2.1 Usage

The influence of Web Components is ubiquitous. Nowadays, almost any popular sites consume web components, with Google being the leader in using web components in their products, such as YouTube Web, YouTube Gaming, Google Earth, and Google Music. Plenty of other famous companies have their websites running with custom components, such as McDonald's, Bloomberg, GitHub, and Victoria's Secret (Polymer 2018i).

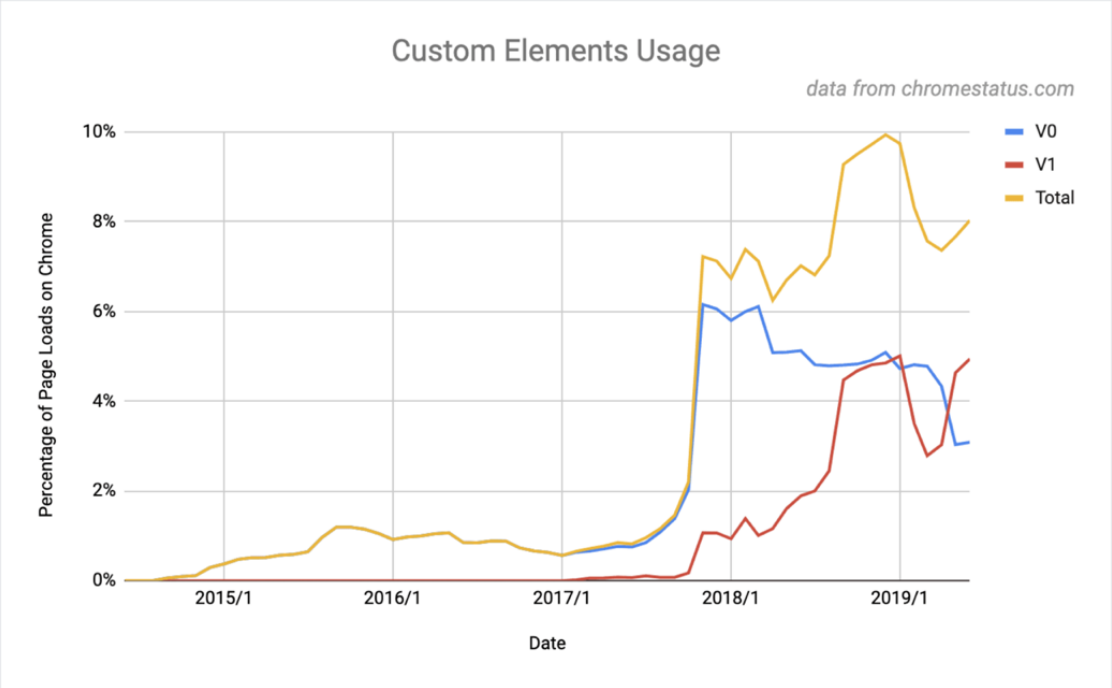


Figure 2. Custom Elements usage (Evans 2019).

Between 5% and 8% of all page load today use web components, making it one of the most successful new web platform features (Evans 2019).

2.2.2 Browser support

When Web Components were first introduced in 2011, all browser vendors were aware of its potential and capability. Various discussions were held between the vendors to agree on how they would implement the standards (Niwa 2019, Wytrebowicz 2018). It took 6 years to see some positive flags in the progress of adopting the standards.

Browser support	CHROME	OPERA	SAFARI	FIREFOX	EDGE
TEMPLATES	STABLE	STABLE	STABLE	STABLE	STABLE
CUSTOM ELEMENTS	STABLE	STABLE	STABLE	POLYFILL DEVELOPING	POLYFILL CONSIDERING
SHADOW DOM	STABLE	STABLE	STABLE	POLYFILL DEVELOPING	POLYFILL CONSIDERING
<SCRIPT TYPE="MODULE">	STABLE	STABLE	10.1	FLAG IN 54	FLAG IN 15
HTML IMPORTS	STABLE	STABLE	POLYFILL ON HOLD	POLYFILL ON HOLD	POLYFILL CONSIDERING

Figure 3. Browser support for Web Components in 2017 (webcomponents.org 2019).

From Figure 3, it is evident that Chrome, Opera, and Safari were the quickest ones to enable the technologies while Firefox and Edge are slowly catching up. For using the standards in a browser that does not support it, a polyfill is needed.

Browser support	CHROME	OPERA	SAFARI	FIREFOX	EDGE
HTML TEMPLATES	STABLE	STABLE	STABLE	STABLE	STABLE
CUSTOM ELEMENTS	STABLE	STABLE	STABLE	STABLE	POLYFILL DEVELOPING
SHADOW DOM	STABLE	STABLE	STABLE	STABLE	POLYFILL DEVELOPING
ES MODULES	STABLE	STABLE	STABLE	STABLE	STABLE

Figure 4. Browser support for Web Components in 2019 (webcomponents.org 2019).

By 2019, every modern browser except for Edge has finished implementing the full set of standards. Nonetheless, Edge status has changed from *Considering* to *Developing*

compared with the 2017 status update from Figure 3. It is detectable that HTML Imports has been replaced by ES Modules, as mentioned in section 2.1.

2.3 Web Components libraries

Web Components standards are web specifications, which means that developers would need to write extremely complicated code to create a native web component. Hence, many Web Components libraries, such as Polymer, X-tag, StencilJS, were created to make it easier and faster to deliver reusable web components (Somesh 2016).

Among those libraries, Polymer was the most popular one since it came from Google. By 2017, there are already two versions of Polymer, namely version 1.x and version 2.x. Most components were written using Polymer libraries because of their constant releases and satisfying quality.

Figure 5. Google Releases Polymer 1.0 (Cimpanu 2015).

Figure 6. Google releases Polymer 2.0 (Polymer 2018h).

2.4 Webcomponents.org

Webcomponents.org allows people to publish and share their components. At that time, it was the only site to provide a suitable environment for registering and showcasing web components.

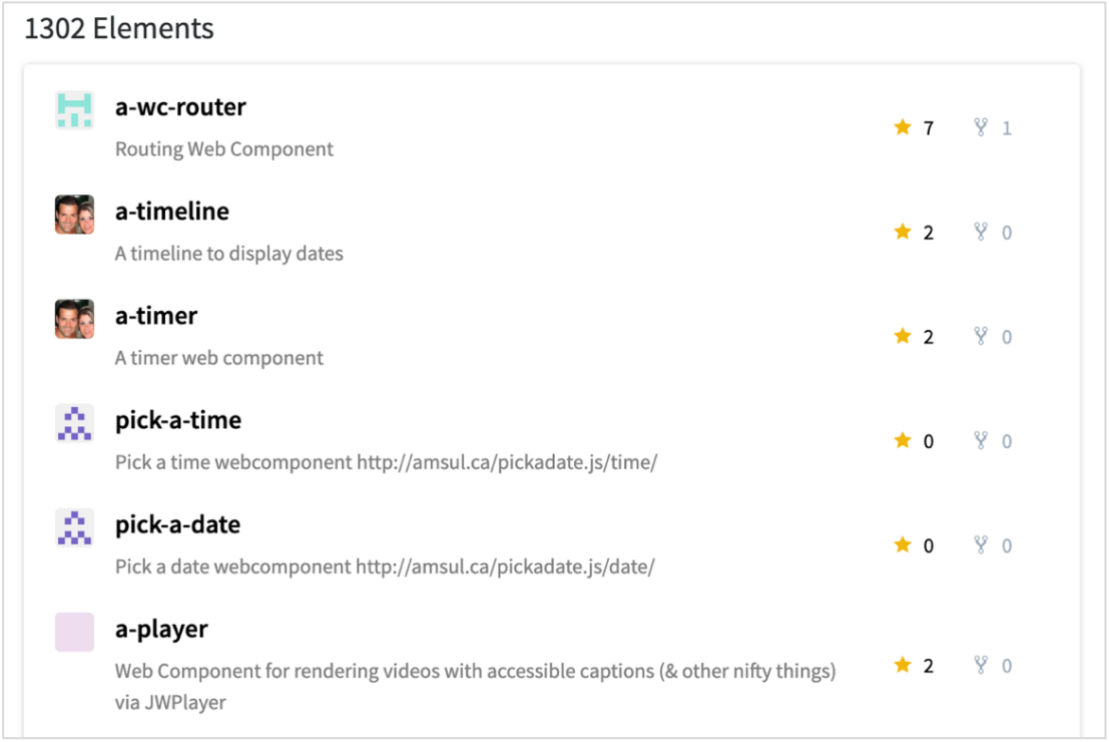


Figure 7. Browsing web components on webcomponents.org in 2017.

Some of webcomponents.org main features are:

- Documentation – presenting the full anatomy of a component if it follows a specific syntax (Polymer 2018e). This feature is similar to Java having Javadoc.
- Content Delivery Network (CDN) – providing a distributer server service for each component's files and assets.
- Inline demo – allowing publishers to make a live demo for their web components (Polymer 2019).
- API – giving people the ability to fetch information about web components from webcomponents.org (WebComponents 2017).

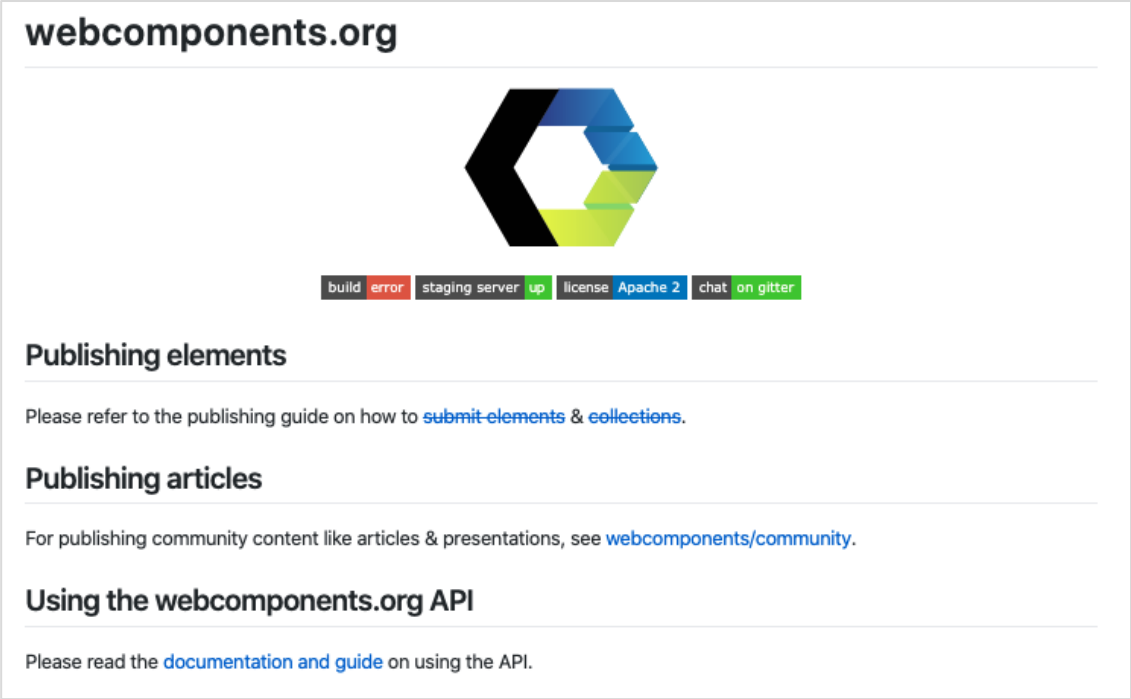


Figure 8. webcomponents.org in GitHub (WebComponents 2016).

All of the mentioned features above are critical to the testing process. The documentation helps to understand better how the component is defined, while the CDN provides a way to online test the component. Lastly, the inline demo gives a quick preview of how the component looks like, and the API assists in exporting web components from the website. Chapter 0 and Chapter 4 will delineate how they assist in the testing process.

3 THE TRIANGLE BETWEEN VAADIN, VAADIN DIRECTORY, AND WEB COMPONENTS

4 AS MENTIONED IN THE

Introduction, Vaadin first wants to integrate Web Components and then imports the functioning components into Vaadin Directory. This chapter discusses the three main entities in this publication, namely Vaadin, Vaadin Directory, and Web Components, along with their relations with each other.

4.1 Vaadin and Vaadin framework

Vaadin Ltd. is a Turku-based tech company founded in 2000 that shares the same name with its main product, Vaadin framework. The Vaadin framework is an open-source Java web framework created to build web applications in the Java language (Vaadin 2019b).

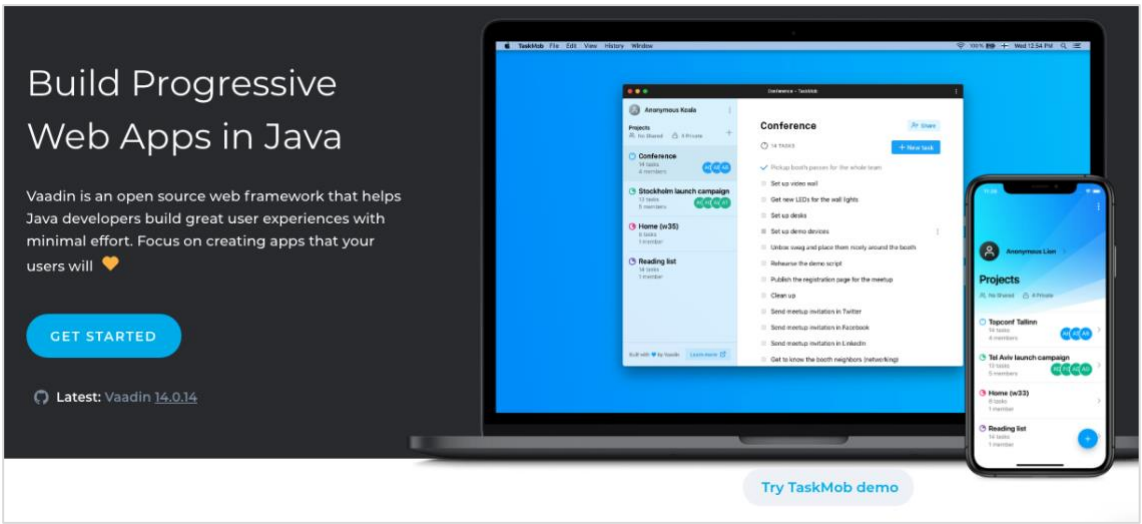


Figure 9. Vaadin main website vaadin.com (Vaadin 2019).

4.2 Reasons why Vaadin integrate Web Components

Java was a dominating language back in the day as a server-side technology in many applications: Android apps, web apps, software, server apps (Paul 2017). However, Java's position in the market share started to decline when Node.js entered the scene in 2009. By 2019, the number of websites built on Node.js has surpassed the Java ones (SimilarTech 2019). As Vaadin framework offers people the ability to use Java to build a full-stack application, it has been an excellent alternative for people who want to avoid JavaScript, and that is how Vaadin gets its reputation in the community.

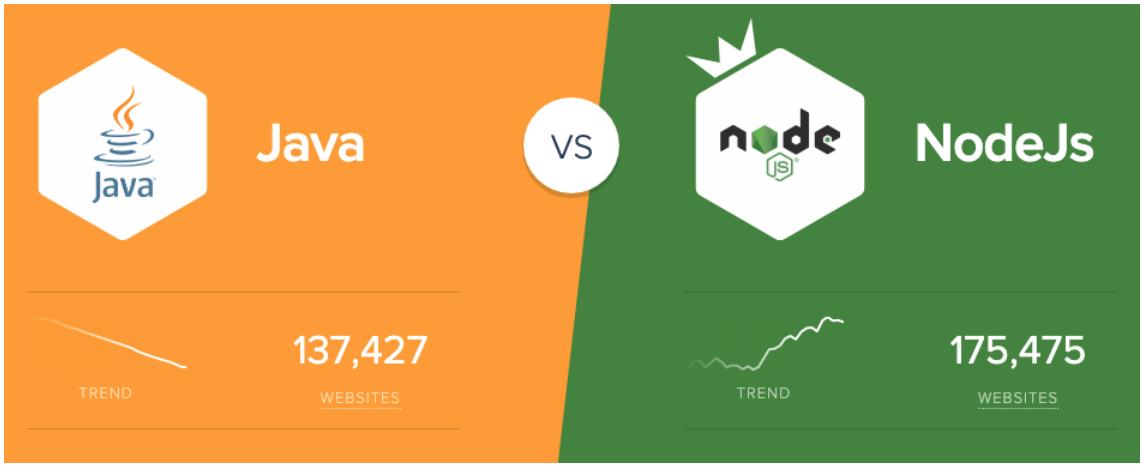


Figure 10. Java vs. Node.js technologies market share (Bui 2019).

Since Web Components standards were getting more and more popular each year, Vaadin needs to adapt as well to stay relevant in the scene. Vaadin understands that sooner or later, they have to integrate the technologies. Furthermore, supporting Web Components would assist Vaadin in reaching more audiences and staying relevant in the web framework scene.

Figure 11. Vaadin CEO presents how Vaadin would leverage Web Components in DevNexus 2015 (Lehtinen 2015).

When Web Components standards became a web specification draft, Vaadin re-wrote the whole framework from Vaadin 8 to Vaadin 10 to support using Web Components inside the framework (Wang 2018). The web components testing process got carried out shortly after that important change.

4.3 The role of Vaadin Directory

Introduced in 2010, Vaadin Directory acted as a platform for distributing and sharing add-on components to be used within the Vaadin framework (Lehtinen 2010b). It consists of hundreds of Java components under many different categories. Vaadin Directory is similar to webcomponents.org to an extent as it offers a community for developers to publish and share their components.

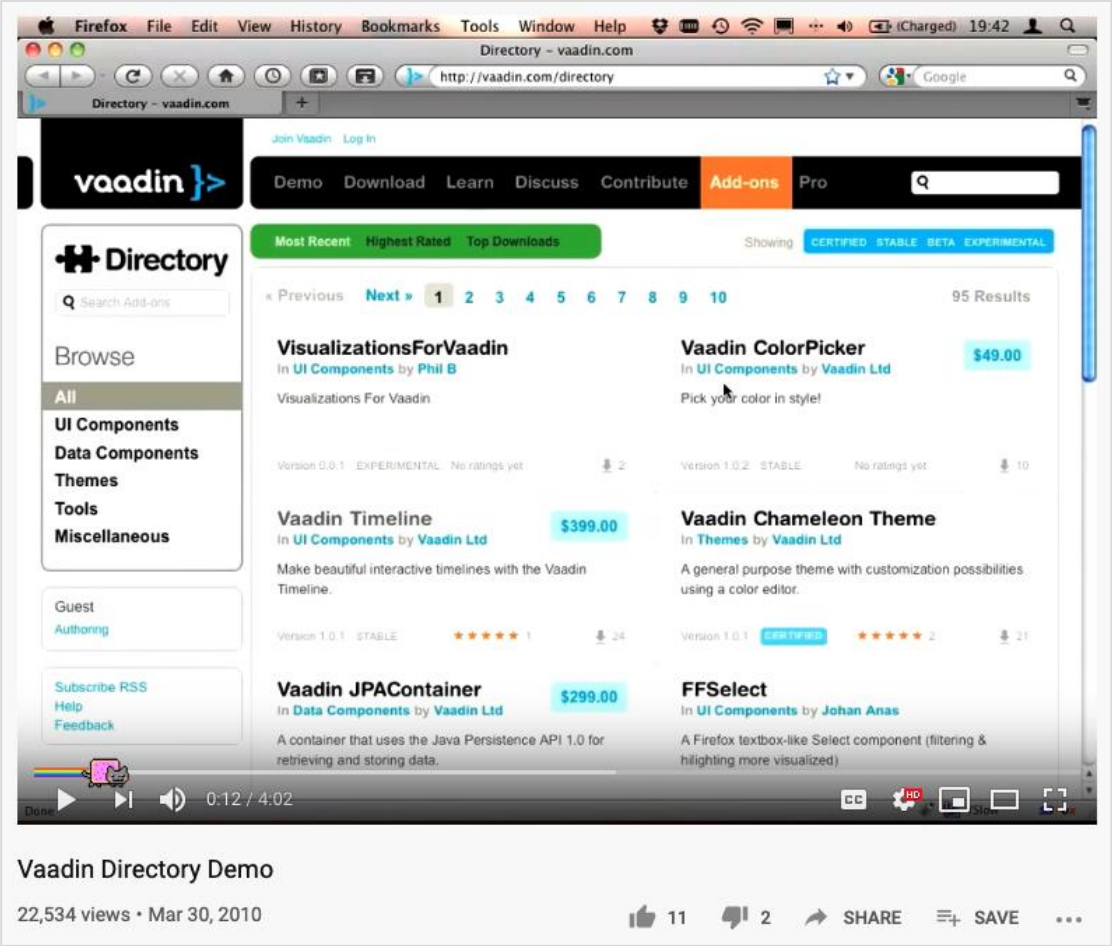


Figure 12. Vaadin Directory demo introduction by Vaadin CEO (Lehtinen 2010a).

Due to its vast growing number of Java components, Vaadin Directory has always been successful in attracting more evaluators for the framework (Pöntelin 2015). Since the new version of Vaadin has utilized the Web Components standards, Vaadin Directory was a critical channel for catalyzing the users' adoption of the new feature.

In summary, Vaadin Directory would need to have a collection of quality web components readily imported and support developers in publishing and maintaining web components.

By completing these two tasks, it will ensure people's interest in the new release and encourage them to use it.

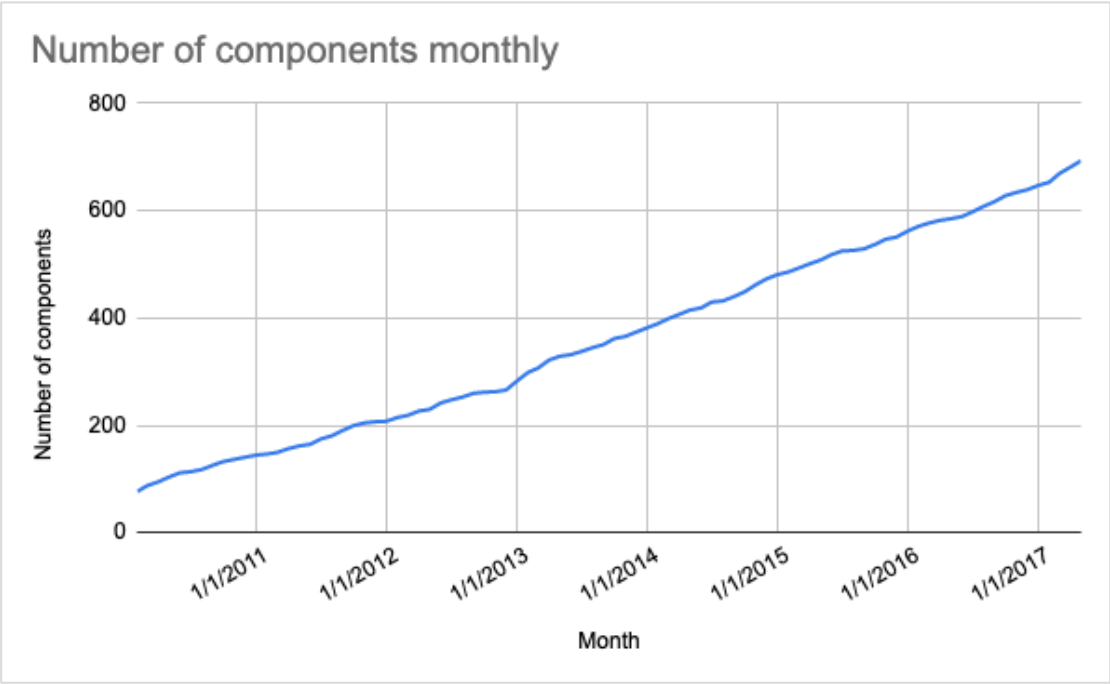


Figure 13. The total amount of components in Directory monthly (Ekblad 2018b).

5 USAGE SHARE OF WEB COMPONENTS LIBRARIES

The usage share of Web Components libraries is the percentage of the web components that use a specific library. Knowing the usage share of the libraries can have a substantial benefit in evaluating the test scope and test implementation.

5.1 Using bower.json to check the library

Before developers can publish their web components to webcomponents.org, they need to publish it first to Bower. Hence, they need to follow the Bower standard by having a bower.json file in the package. Bower.json is a configuration file for the package, which is a common requirement among package managers.

Since bower.json includes all the dependencies that the package is using, it can be quickly checked the file to see which Web Components library the package depends on.

26 lines (26 sloc) | 531 Bytes

```
1  {
2    "name": "iron-timer",
3    "authors": [
4      "collaborne"
5    ],
6    "description": "A simple countdown timer",
7    "main": "iron-timer.html",
8    "keywords": [
9      "web-components",
10     "polymer",
11     "iron",
12         "timer"
13   ],
14   "license": "Apache-2.0",
15   "homepage": "https://github.com/collaborne/iron-timer",
16   "ignore": [
17     "**/.*"
18   ],
19   "dependencies": {
20     "polymer": "Polymer/polymer#^2.0.0"
21   },
22   "devDependencies": {
23     "iron-component-page": "PolymerElements/iron-component-page#^2.0.0",
24     "web-component-tester": "*"
25   }
26 }
```

Figure 14. An example of a bower.json file (Collaborne 2015).

It is observable from the bower.json file that the web component uses the library Polymer 2. Henceforth, this method can be applied to check the web component library each component is using.

5.2 Code implementation

This thesis uses a Python script to detect the web component library of each web component. Small chunks of code are provided in this section to help clarify the implementation.

Before writing the script, a manual process of looking at 10 components bower.json results in Polymer being the dominating library, with 9 out of 10 use either Polymer 1 or Polymer 2. Therefore, the script is going to check first if the component uses Polymer 1 or Polymer 2, then the next step will be defined according to the obtained result.

```
# Link to the raw data of bower.json file
bowerJsonUrl = "https://raw.githubusercontent.com/vladimirbrasil/a-timer/master/bower.json"

# Get the JSON from the URL
response_data = urlopen(url).read()
json_data = simplejson.loads(response_data)

# Define the regex for checking Polymer 1 & 2
polymer1Regex = r"(\^1)|(\-\s*1)|([^.]1\.)|(^1.)"
polymer2Regex = r"(\^2)|(\-\s*2)|([^.]2\.)|(^2.)"

# Check if Polymer is in the dependencies
if 'polymer' in json_data ['bower']['dependencies']:
    # Get the polymer version
    polymerVer = json_data['dependencies']['polymer']
    if re.search(polymer2Regex, polymerVer):
        returns "Polymer 2"
    elif re.search(polymer1Regex, polymerVer):
        returns "Polymer 1"
else:
    returns "Other"
```

Figure 15. Detecting Web Components library from bower.json.

From Figure 15, the process begins by using urllib and simeplejson to parse the JSON from the bower.json URL. Then, two Regular Expressions (regex) for Polymer 1 and 2 were created to catch any variation of Semantic Versioning (SemVer) in the bower.json file. Finally, an if/else statement is written to check the regex against the obtained library from bower.json.

5.3 Usage share of Web Components libraries

The statistics were obtained from the script in section 4.2. Out of 1302 components, 696 use Polymer 2, 504 use Polymer 1, and the rest use other libraries. The Polymer library accounts for more than 90% of the usage share. As a result, Polymer would have a prominent effect on the testing process, which will be mentioned in section 6.2.

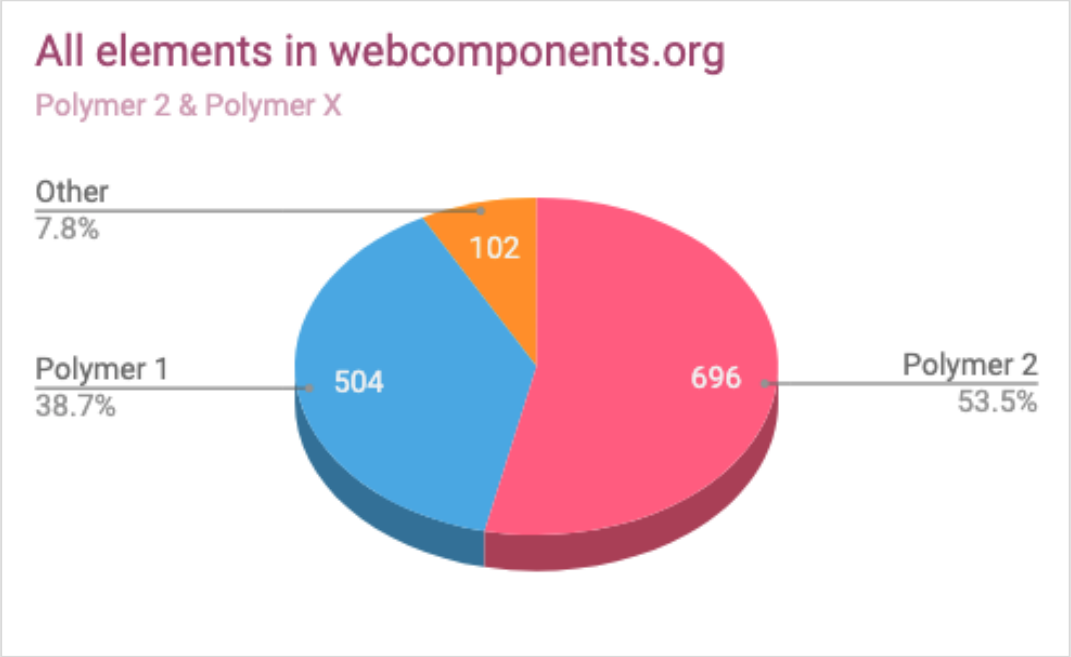


Figure 16. Usage share of web components libraries used.

6 PREREQUISITES FOR TESTING

This chapter describes the prerequisites for the web components testing process. The prerequisites include storing web components information, preparing an online testing environment, and collecting intelligence on the Polymer library. This thesis uses Python to write scripts for working with the data and the API.

It is worth mentioning that at the time when this testing process happened, there was no other similar work that is available in any publication, article, or blog by searching the term ‘Test hundreds of web components’ on Google. While there were multiple articles about quality testing a single web component, to test hundreds of web components is an entirely different subject. Therefore, all the prerequisites work for the testing process were defined manually by the author and not following any standards.

6.1 Storing web components data

As the number of web components from webcomponents.org exceeds more than one thousand, a storage of data is essential for keeping all the details and testing results of the components. Therefore, a spreadsheet has been created to make it easier to extract and input information.

6.1.1 Using webcomponents.org API

The first task is to import every component in webcomponents.org into the spreadsheet along with their information. As mentioned in section 2.4, webcomponents.org comes with a basic set of APIs which allows users to fetch data from their site in a JavaScript Object Notation (JSON) response (WebComponents 2017). The endpoint /search is the primary API call which is used to fetch every web component in the site. It returns a list of maximum 20 components at once. Since there is a limit, the code loops between exporting the current list of components and getting the next one.

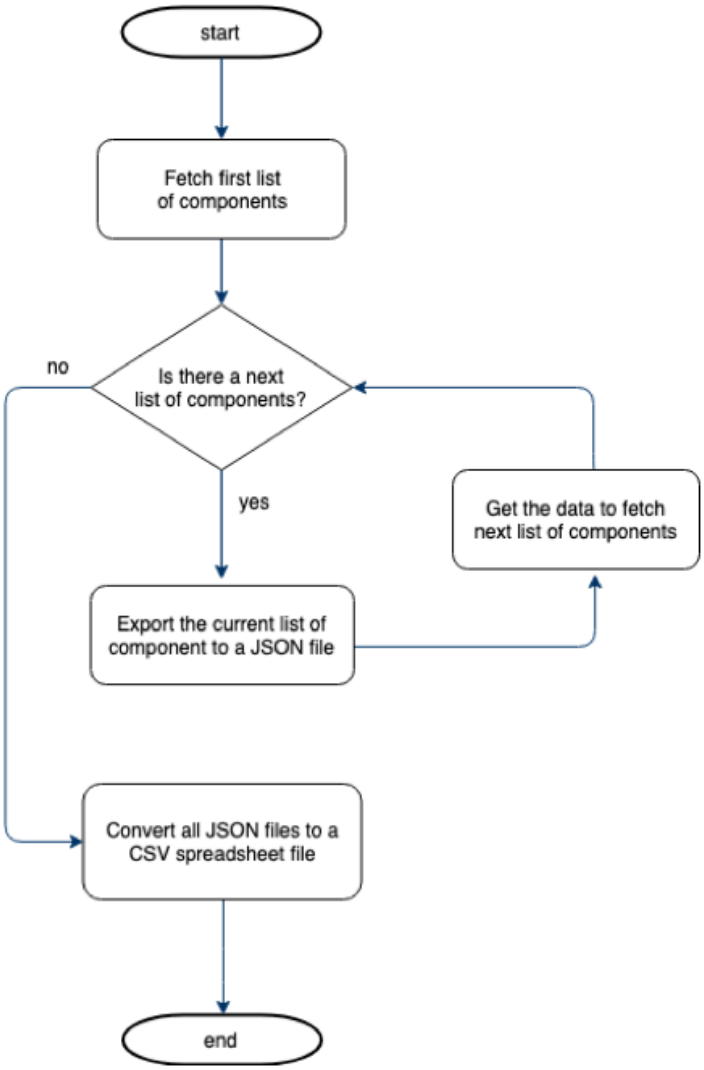


Figure 17: Fetching components from webcomponents.org flowchart.

```
https://www.webcomponents.org/api/search/kind:element%20kind:element?limit=20&count
```

Figure 18. webcomponents.org /search API full URL.

In Figure 18, the part *kind:element%20kind:element* tells the server to return all components from the site, the parameter *limit=20* sets the number of returned components, and the parameter *count* tells the server to return the total number of components in the response.

```
"cursor":
    "False:Cq8DCq4BCosBgP_-9w6MJMjwnN_8jIGcioyLkJLSmpOakpqRi4z_AHRtoKCZi4ygoP8AXZ6Pj5qRmJaRmv8Ac
    3RtlpGbmof_AF2Nmo-Q_wBzdG2bkJyglpv_AF2Jnp6blpHQiZ6em5aR0ouah4vSmZaak5v_AHN_iZ6em5aR0ImenpuWk
    dKLmoeL0pmWmpOb_wD__hAoIQTdBJAdPBsZUABaCwkNUZFWyZQkdxADYMqbx7EHEg1Eb2N1bWVudEluZGV4GrYBKEFOR
    CAoSVMgImN1c3RvbWVyX25hbWUiICJhcHBlbmdpbmUiKSAoSVMgImdyb3VwX25hbWUiICJzfmN1c3RvbS1lbGVtZW50c
    yIpIChJUyAibmFtZXNwYWNlIiAiIikgKElTICJpbmRleF9uYW1lIiAicmVwbyIpIChPUiAoUVQgImVsZW1lbnQiICJyd
    GV4dF9raW5kIikgKElTICJyYXRvbV9raW5kIiAiZWxlbWVudCIpKSk6GQoMKE4gb3JkZXJfaWQpEAEZAAAAAAAA8P9KG
    QgBOhJidGlfZ2VuZXJpY19zY29yZXJA6Ac",
"count": 2183,
"results": [
    {
        "description": "@lrnwebcomponents Monorepo for NPM based element definitions",
        "avatar_url": "https://avatars3.githubusercontent.com/u/12715666?v=4",
        "updated_at": "2019-11-29T19:35:30Z",
        "npmFullPackage": "@lrnwebcomponents/iconset-demo",
        "repo": "lrnwebcomponents",
        "subscribers": 22,
        "owner": "elmsln",
        "kind": "element",
        "npmPackage": "iconset-demo",
        "version": "2.2.0",
        "stars": 75,
        "forks": 34
    },
```


Figure 19. An example of webcomponents.org API response.

From the response in Figure 19, there are three pairs of key names and key values. The key name *cursor* has the data for getting the next list of components. Then, the key name *count* shows how many components there are, and finally, the key name *results* returns the list of components along with their information.

6.1.2 Code implementation

This section manifests the Python code for getting the usage share of Web Components libraries. From the flowchart in Figure 17, the work can be divided into three parts: fetching the first list of web components from webcomponents.org, looping to get all the lists and export them to JSON and finally, combine all the JSON to a CSV spreadsheet.

For fetching the first list of web components, the API gets called one time, so the first *cursor* and *results* data can be read.

```
# Define the base URL (wc.org API https://github.com/webcomponents/webcomponents.org/blob/ma
base = "https://www.webcomponents.org/api/search/kind:element%20kind:element?limit=20&count"
currentURL = base

# Load first JSON to get the total elements number
json_data = get_json_from_url(currentURL)

# Get value of cursor for the loading of next JSON
nextCursor = json_data["cursor"]
```

Figure 20. Fetching the first list of web components from webcponents.org.

The key *cursor* works so that every response returns a value that can be used to fetch the next list of components. Since an absence of this key name indicates there are no more components, a *while* loop with the condition that key *cursor* is used to loop between getting the component list and exporting it to a JSON file.

```
while nextCursor is not None:
    # Name the file with numering start from 01
    # elementsJSON_path is the physical path to store the file
    JSONFilePath = "{0}{1}".format(elementsJSON_path, "%02d.json")

    # Create & open the file, then export the results
    with io.open(JSONFilePath, 'w', encoding='utf-8') as f:
       f.write(simplejson.dumps(json_data['results'], ensure_ascii=False))

     # URL encode the cursor value
   nextCursorValue = urllib.parse.quote(nextCursor)

   # Generate the cursor parameter for the URL
   currentURL = "{0}&cursor={1}".format(base, nextCursorValue)

   # Read the next JSON
   json_data = get_json_from_url(currentURL)

   # Get value of cursor for the loading of next JSON
   nextCursor = json_data["cursor"]

   # Increment the counter value for the naming
   counter += 1
```

Figure 21. Looping to get the next web component list and exporting it to JSON.

The loop begins with opening a new JSON file and then dumping the current value of the key *results* to the file. With this method, every list of 20 components in each response is saved to a JSON file. Then, the program gets the *cursor* value and use it to formulate the next URL for the API call.

When all the components have been stored locally, the script will combine all of those JSON files into a single CSV spreadsheet file. For this purpose, this thesis uses the library pandas.

```
# Create a panda data frame
frame = pd.DataFrame()

# Load all JSON files in the specific directory
for file in glob.glob(elementsJSON_path + "*.json"):
    pandaObj = pd.read_json(file, encoding='utf-8')
    frame = frame.append(pandaObj, ignore_index=True)

# Exporting to a CSV file with custom defined columns
frame.to_csv('{0}/allElements.csv'.format(elementsCSV_path),
index=True, columns=['repo', 'owner', 'github', 'wc_url', ...])
```

Figure 22. Using pandas to combine all JSON files to a single CSV spreadsheet.

The process begins by creating a DataFrame, a two-dimensional size-mutable data structure in pandas (pandas 2019)**;** then, appending all stored JSON files into the frame and finally, exporting to a CSV file with custom-defined columns. Custom-defined columns refer to the part ['repo', 'description', 'owner', …].

	A	B	D	E	F
1	**repo**	**owner**	**github**	**wc**	**vd**
2	adebray-turing	Zleub	https://github.cc	https://ww	https://
3	animation-keyframes	valdrinkoshi	https://github.cc	https://ww	https://
4	apod-image	gthmb	https://github.cc	https://ww	https://

Figure 23. Spreadsheet custom-defined columns.

For custom columns, if there is a *repo* field from the JSON response, then the DataFrame will use that data to populate the *repo* column. Otherwise, if there is no *repo* field from the response then the cell value will be empty. The purpose of this is to have columns for extra user-input information.

Subsequently, a file name allElements.csv is created, which contains all the web components in webcomponents.org with their information.

	A	B	D	E	F	I	J	K	L	M	N	O	P	Q	R
1	repo	owner	github	wc	vd	screens	samplec	supp	rat	comment	pc	te	i	categ	bowe
7	app-localize-behavior	PolymerElements													
8	app-route	PolymerElements													
9	app-section-head	PolymerAppElements													
10	app-toast	jifalops													
11	app-toasts	jifalops													
12	battery-status-icon	Protoss78													
13	bender-clock	itsMeBender													
14	bowser-polymer	LssPolymerElements													
15	butter-github-star	JCrestel													
16	button	BrightspaceUI													
17	candy-progress	Collaborne													
18	carrousel-list	itsMeBender													
19	checkbox	BrightspaceUI													
20	chemical-element-visualisatio	FlorianFe													
21	code-sample	kcmr													
22	colors	BrightspaceUI													

Figure 24. The output spreadsheet allElements.csv.

6.1.3 Populating extra information

Figure 24 signifies that the spreadsheet has stored all the web components' repository and their respective owners. Since the links to their GitHub repositories and webcomponents.org are needed later in the testing process, they will be populated in the

spreadsheet. This task can quickly be achieved by using the CONCATENATE function of the spreadsheet. The next part explains more about the function.

For column D which stores all the GitHub URLs, input this formula:

=CONCATENATE("https://github.com/", B2, "/", A2)

And for column E which stores all the webcomponents.org URL, input this formula:

=CONCATENATE("https://www.webcomponents.org/element/", B2, "/", A2)

After applying the formula to the whole column, every cell is filled with the according URLs pointed to each web component page on GitHub and webcomponents.org.

	A	B	D	E
1	repo	owner	github	wc
2	adebray-turing	Zleub	https://github.com/Zleub/adebray-turing	https://www.webcomponents.org/element/Zleub/adebray-turing
3	animation-keyframes	valdrinkoshi	https://github.com/valdrinkoshi/animation-k	https://www.webcomponents.org/element/valdrinkoshi/animation-k
4	apod-image	gthmb	https://github.com/gthmb/apod-image	https://www.webcomponents.org/element/gthmb/apod-image
5	app-datepicker	motss	https://github.com/motss/app-datepicker	https://www.webcomponents.org/element/motss/app-datepicker
6	app-layout	PolymerElements	https://github.com/PolymerElements/app-la	https://www.webcomponents.org/element/PolymerElements/app-la
7	app-localize-behavior	PolymerElements	https://github.com/PolymerElements/app-lc	https://www.webcomponents.org/element/PolymerElements/app-lo
8	app-route	PolymerElements	https://github.com/PolymerElements/app-rc	https://www.webcomponents.org/element/PolymerElements/app-ro
9	app-section-head	PolymerAppElemer	https://github.com/PolymerAppElements/ap	https://www.webcomponents.org/element/PolymerAppElements/ap
10	app-toast	jifalops	https://github.com/jifalops/app-toast	https://www.webcomponents.org/element/jifalops/app-toast
11	app-toasts	jifalops	https://github.com/jifalops/app-toasts	https://www.webcomponents.org/element/jifalops/app-toasts
12	battery-status-icon	Protoss78	https://github.com/Protoss78/battery-status	https://www.webcomponents.org/element/Protoss78/battery-status-
13	bender-clock	itsMeBender	https://github.com/itsMeBender/bender-clo	https://www.webcomponents.org/element/itsMeBender/bender-cloc
14	bowser-polymer	LssPolymerElemen	https://github.com/LssPolymerElements/bo	https://www.webcomponents.org/element/LssPolymerElements/bo
15	butter-github-star	JCrestel	https://github.com/JCrestel/butter-github-st	https://www.webcomponents.org/element/JCrestel/butter-github-sta
16	button	BrightspaceUI	https://github.com/BrightspaceUI/button	https://www.webcomponents.org/element/BrightspaceUI/button
17	candy-progress	Collaborne	https://github.com/Collaborne/candy-progre	https://www.webcomponents.org/element/Collaborne/candy-progre
18	carrousel-list	itsMeBender	https://github.com/itsMeBender/carrousel-li	https://www.webcomponents.org/element/itsMeBender/carrousel-li
19	checkbox	BrightspaceUI	https://github.com/BrightspaceUI/checkbox	https://www.webcomponents.org/element/BrightspaceUI/checkbox
20	chemical-element-visualisatio	FlorianFe	https://github.com/FlorianFe/chemical-elem	https://www.webcomponents.org/element/FlorianFe/chemical-elem
21	code-sample	kcmr	https://github.com/kcmr/code-sample	https://www.webcomponents.org/element/kcmr/code-sample
22	colors	BrightspaceUI	https://github.com/BrightspaceUI/colors	https://www.webcomponents.org/element/BrightspaceUI/colors

Figure 25. The spreadsheet with populated information.

6.2 Preparing the online testing environment

One can offline test a web component by installing it from Bower with a command (e.g., bower install –save vaadin/vaadin-button). Offline testing is easy to achieve, albeit rather time-consuming and not suitable in this testing process. To online test a component, there are two things needed: a way to import web components online and an online code editor platform to test the web components implementation.

Having mentioned in section 2.4, one of webcomponents.org main features is offering its users a CDN. Every web component in webcomponents.org is available on the CDN as

a full package. Then, the CDN provides users with a Uniform Resource Locator (URL) to the package files. With this link, the browser can import a web component online by using the HTML Imports standard, which is mentioned in section 2.1.

```
<link rel="import" href="https://raw-dot-custom-elements.appspot.com
/samuelli/progress-bar/v2.0.1/progress-bar/progress-bar.html">
```

Figure 26. Importing a web component by using a CDN and HTML Imports.

This thesis uses CodePen as the online code editor platform for the testing process. CodePen makes it easy to write codes online, import libraries from different sources, and share snippets with other people. Additionally, CodePen allows users to create a collection of code snippets, a feature that can be used to create a collection of web components demo.

CodePen User Interface (UI) has four different windows: an HTML editor window, a Cascading Style Sheets (CSS) editor window, a JavaScript editor window, and a live preview.

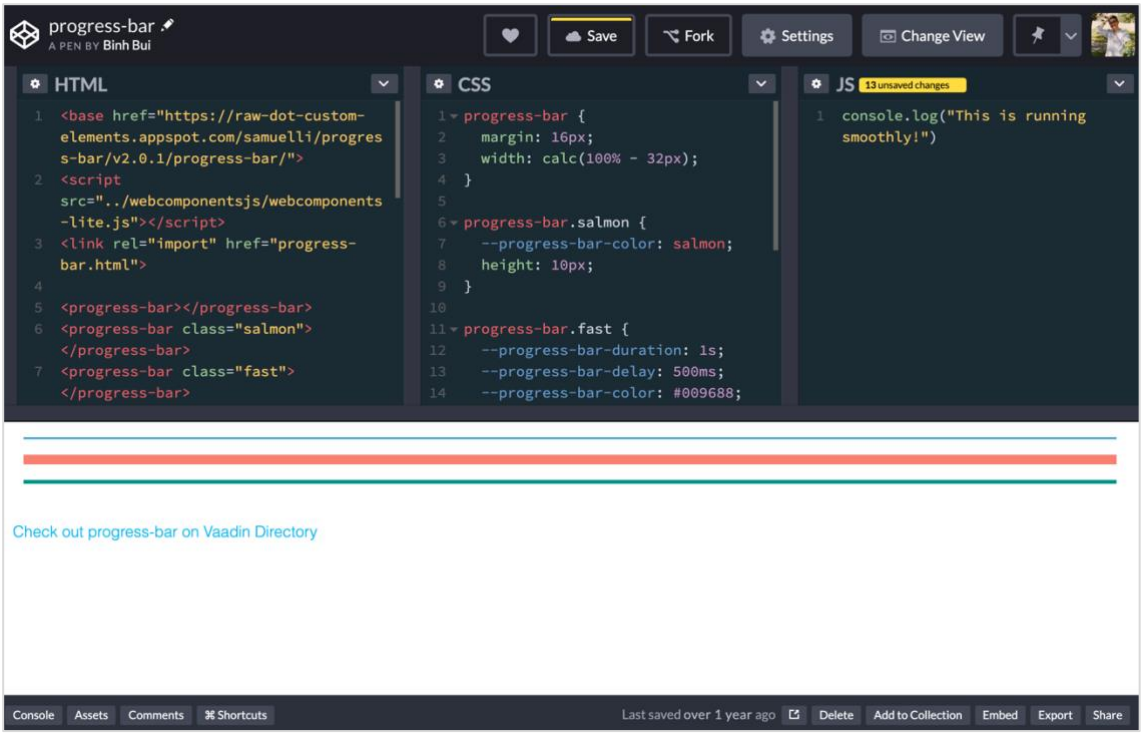


Figure 27. An example of testing a web component on CodePen.

Figure 27 demonstrates how to use the webcomponents.org CDN to write and test a component. The visual result is observable in the live preview window. The next paragraphs explain more about the snippet code.

```
<base href="https://raw-dot-custom-elements.appspot.com/samuelli/
progress-bar/v2.0.1/progress-bar/">
<script src="../webcomponentsjs/webcomponents-lite.js"></script>
<link rel="import" href="progress-bar.html">

<progress-bar></progress-bar>
<progress-bar class="salmon"></progress-bar>
<progress-bar class="fast"></progress-bar>
```

Figure 28. HTML code of a web component demo on CodePen.

In the HTML code, a base tag is defined, which acts as the base for all the relative URLs in the document (MDN 2019a). Since all the imports are from a single package, they have the same URL beginning. Then, the polyfill *webcomponents-lite.js* and the web component file *progress-bar.html* are imported. As mentioned in section 2.2.2, a polyfill is a piece of code that implements a feature that the browser has not supported. Finally, three instances of the custom element are defined.

```
progress-bar {
  margin: 16px;
  width: calc(100% - 32px);
}

progress-bar.salmon {
  --progress-bar-color: salmon;
  height: 10px;
}

progress-bar.fast {
  --progress-bar-duration: 1s;
  --progress-bar-delay: 500ms;
  --progress-bar-color: #009688;
  height: 4px;
}
```

Figure 29. CSS code of a web component demo on CodePen.

Figure 29 shows that the component progress-bar acts as a standard HTML element in the CSS.

6.3 Learning the Polymer library

From section 5.3, the obtained statistics proved that the Polymer library was a prevalent choice for developing a web component. As a result, getting a satisfying amount of knowledge in the library was crucial to the testing case.

Although the library involves lots of features and specifications, there are several key features which are: element defining, properties declaring, property to attribute reflecting, and data binding. Understanding those key features helps to make more test cases in the testing process.

6.3.1 Element defining

Custom elements specification allows a whole new element with a custom name to be defined (Polymer 2018b).

```
class MyPolymerElement extends Polymer.Element {
  …
}
customElements.define('my-polymer-element', MyPolymerElement);
```

Figure 30. Defining an element in Polymer.

By specification, the custom element's name needs to start with a lower-case letter and includes a dash (-) (Polymer 2018).

It is also possible to extend an existing element by merely making a subclass to the original element. The ability to subclass opens room for adding new subsets of features and gives the end-users plenty of choice.

```
// Subclass existing element
class MyElementSubclass extends MyPolymerElement {
   ...
}
// Register custom element definition using standard platform API
customElements.define(MyElementSubclass.is, MyElementSubclass);
```

Figure 31. Extending an existing custom element in Polymer.

6.3.2 Properties declaring

Properties declaring is an act in which one declares new custom properties for the component to enable various features in the data system (Polymer 2018d). Polymer allows users to declare normal properties or more advanced properties, namely computed properties.

```
class XCustom extends PolymerElement {
  static get properties() {
    return {
      user: String,
      isHappy: Boolean,
      count: {
        type: Number,
        readOnly: true,
        notify: true
      }
    }
  }
}
customElements.define('x-custom', XCustom);
```

Figure 32. Declaring normal properties in Polymer.

In this scenario, there are three properties defined: a String named *user*, a Boolean named *isHappy*, a Number named *count* with the read-only option, and property change notification. The property notification works so that whenever the property *count* changes, an event named *count-changed* will be fired so users can listen to that event and define what needs to be done next (Polymer 2018g).

Meanwhile, computed properties are virtual properties computed based on one or more paths (Polymer 2018f). The reason it is called computed is that there is a computing function to assess its final value.

```
class XCustom extends Polymer.Element {

    static get is() {
        return 'x-custom';
    }

    static get properties() {
        return {
            first: String,

            last: String,

            fullName: {
                type: String,
                // when `first` or `last` changes `computeFullName` is called once
                // and the value it returns is stored as `fullName`
                computed: 'computeFullName(first, last)'
            }
        }
    }

    computeFullName(first, last) {
        return first + ' ' + last;
    }
}

customElements.define(XCustom.is, XCustom);
```

Figure 33. Declaring computed properties in Polymer.

The example code has three String properties: *first*, *last*, and *fullName*. The value of fullName is established by the function *computeFullName(first, last),* which takes the concatenation value of *first* and *last* as the parameters.

6.3.3 Property to attribute reflecting

HTML elements have attributes (for example, a div element has attributes like id and class), they change the elements' behavior and configuration based on the users' need (MDN 2019b). Each of the DOM element attributes is mapped to a property in the system. Consequently, when a new property is declared, it will be assigned to a new attribute as well.

Attributes and their according properties may not be in sync with each other. Polymer provides the ability to reflect properties back to attributes by using the setting *reflectToAttribute: true*. This action guarantees the synchronization between the attribute and property value.

```
class XCustom extends Polymer.Element {

  static get properties() {
    return {
      loaded: {
        type: Boolean,
        // Turn on the setting for reflecting
        reflectToAttribute: true
      }
    }
  }

  _onLoad() {
    this.loaded = true;
    // results in this.setAttribute('loaded', true);
  }
}
```

Figure 34. Property to attribute reflecting in Polymer.

This setting is crucial for properties that are continuously updated and needs users noticing. On the other hand, properties that are set one-time only or read-only do not need this setting.

6.3.4 Data binding

A data binding connects data from a host element to a property or attribute of the target element (Polymer 2018c). It binds the data between two sources, either one-way or two-way.

Two-way binding '{{ }}' supports both upward and downward data flow, which means that whenever there is a change in one value, the other changes too. In contrast, one-way binding '[[ ]]' means that the data only goes from the host to the target.

```
// Two-way binding
<my-element my-property="{{hostProperty}}">
<a href$="{{hostProperty}}">

// One-way binding
<target-element name="[[myName]]"></target-element>
<div>[[myName]]</div>
```

Figure 35. Data binding in Polymer.

Notice that for a native attribute, a $ sign needs to be added after the attribute name.

7 TESTING WEB COMPONENTS

The prerequisites work has produced a testing ecosystem, in which a web component can be tested online by using the spreadsheet data; then, the obtained testing result is documented back to the spreadsheet. This chapter describes the testing process, along with its criteria, scope, environment, and implementation.

7.1 Test criteria

Since the nature of this research is quantitative rather than being qualitative, the testing criteria mainly focus on observation and usage problems. The criteria cover factors like the functioning under different browsers, the responsiveness, the functionality, and the category. Here are the questions which are asked when testing each component:

- Does the component function ordinarily on all each browser?
- How long does it take to load the component?
- Is the component responsive under different viewports (mobile, tablet, desktop)?
- How complex is the component usage?
- What category does the component fall under (UI, Data, Theming, Miscellaneous)?

7.2 Test scope

Section 5.1 result clearly shows that Polymer is the dominating library among other libraries. To achieve the most optimal results for Vaadin, the Polymer library was chosen as the testing library. In addition, only components that are using Polymer 2 were selected.

The reason is that when Polymer 2 got released, Polymer 1 became deprecated, and the Polymer team recommends that developers should use Polymer 2 if they want to make a new component. Moreover, as there are no more updates to Polymer 1, developers will upgrade their Polymer 1 components to Polymer 2 to be maintainable.

From section 5.3, there a total of 696 Polymer 2 web components.

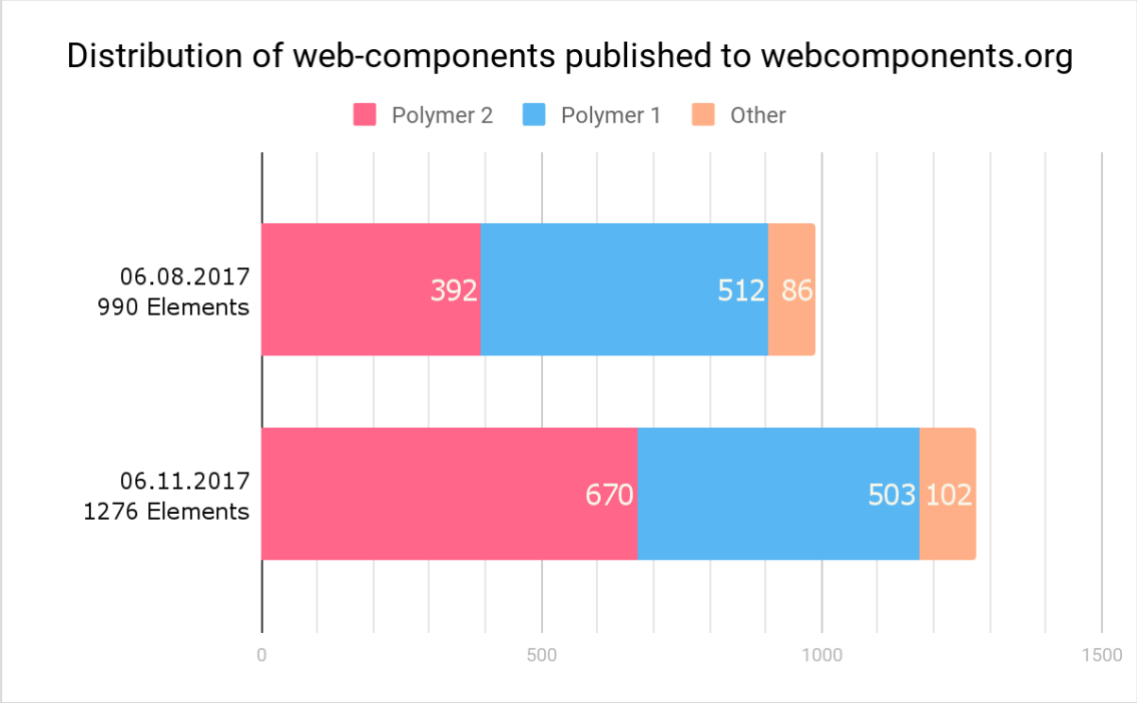


Figure 36. Distribution of the libraries used by web components in webcomponents.org.

From Figure 36, it is noticeable that the number of Polymer 2 components are rising while the number of Polymer 1 components is still the same.

7.3 Test environment

Since web components work on websites and web applications, the primary environment is the browser. All the popular browsers are selected as the testing platforms, which include Chrome, Opera, Safari, Firefox, Microsoft Edge (Edge), and Internet Explorer (IE11).

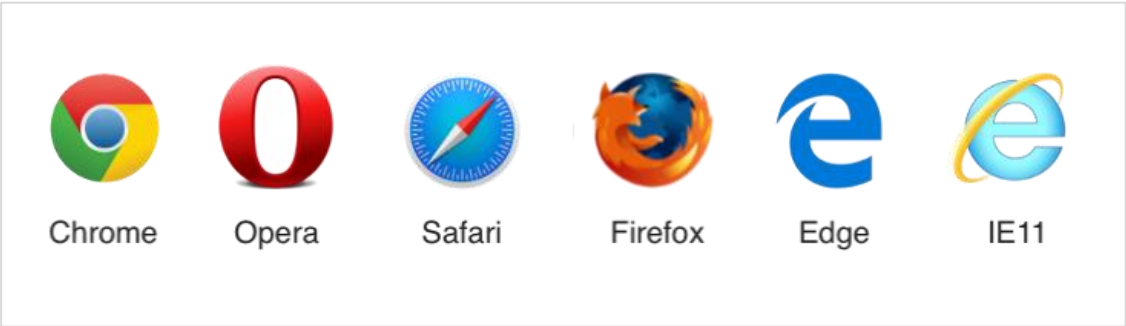


Figure 37. Browsers selected as testing platforms (O'Shaughnessy 2017).

The operating system on which the testing process occurred is macOS 10.13 High Sierra. As a result, all the browsers are macOS applications. Here is the list of browsers used with their version number:

- Chrome Version 59.0.3071.115 (Official Build) (64-bit)
- Opera Version 63.0.3368.107 (64-bit)
- Safari Version 10.1.2 (12603.3.1)
- Firefox 54.0 (64-bit)
- Edge Version 40.15063
- IE11 Version 11.0.10240.16384

As Edge and IE11 are only available in the Windows operating system, VirtualBox was used to access Edge and IE11. VirtualBox is a software that allows users to emulate the Windows operating system in macOS (Stegner 2017).

7.4 Test risk and control

As this is quantitative testing research, the highest priority is to reduce the duration of testing one component as much as possible. All the control solutions to the risks are defined with time being the main factor.

Table 1. Testing process risk with their controls.

Risk	Control
No demo	Check if there is documentation and follow it.
No documentation	Check if the component can be used intuitively, otherwise skip the component.
Not a visual component	Check if there is a demo or documentation. If not, skip the component.
Visual components do not appear	Check quickly if there is a workaround or any problem with the implementation. If it takes too much time, skip the component.

7.5 Test implementation

This section describes how the web components testing was performed. Since this is a repetitive process with each component, a single component is chosen as the test candidate to better visualize and clarify each step of the process. Vaadin-button is chosen to be the candidate here.

Before starting, the spreadsheet and the browsers need to be opened. These tools remain open all the time during the process. Note that IE11 and Edge are opened from VirtualBox.

The process begins by opening the vaadin-button page on webcomponents.org and github.com. The links to these pages have already been populated in section 6.1.3.

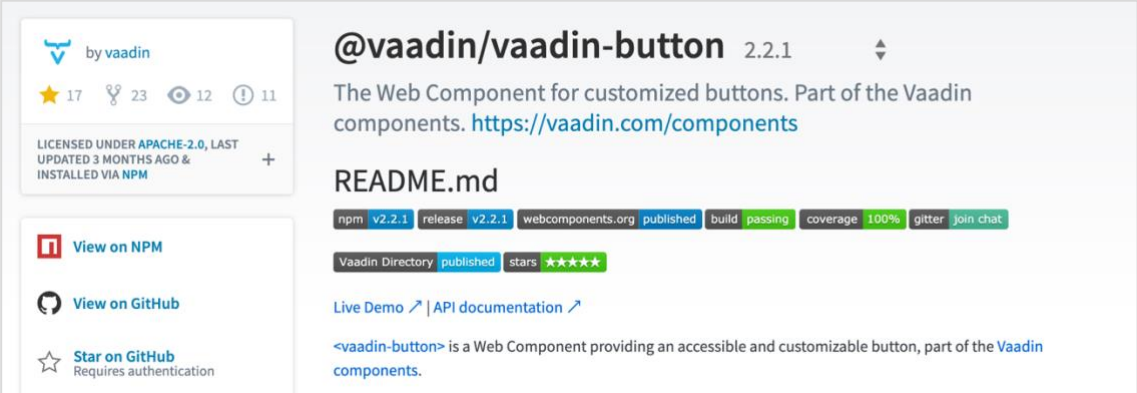


Figure 38. Vaadin-button page on webcomponents.org.

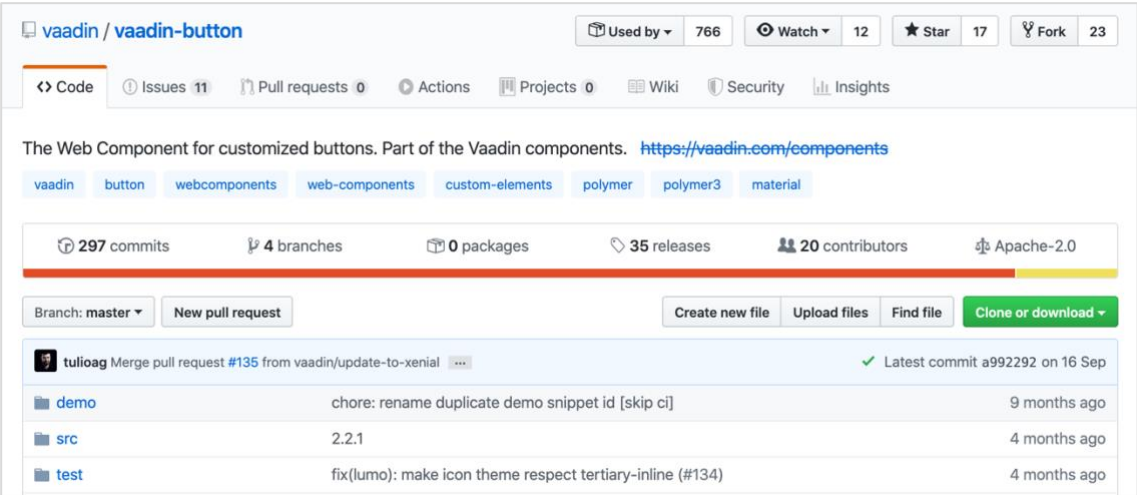


Figure 39. Vaadin-button repository on github.com.

From those pages, a skimming through the description is done to understand briefly what the component is about. Most of the time, the component name suggests the definition of the component. In this case, the custom component is vaadin-button, which is obviously a button component. The skimming process continues by checking the documentation, the inline demo (if present), and the demo file. By looking at the source code of the component, one can see which properties they component has.

```
 * The following attributes are exposed for styling:
 *
 * Attribute | Description
 * --------- | -----------
 * `active` | Set when the button is pressed down, either with mouse, touch or the keyboard.
 * `disabled` | Set when the button is disabled.
 * `focus-ring` | Set when the button is focused using the keyboard.
 * `focused` | Set when the button is focused.
 *
 * See [ThemableMixin – how to apply styles for shadow parts](https://github.com/vaadin/vaadin-themable-mixin/wiki)
 *
 * @memberof Vaadin
 * @mixes Vaadin.ElementMixin
 * @mixes Vaadin.ControlStateMixin
 * @mixes Vaadin.ThemableMixin
 * @mixes Polymer.GestureEventListeners
 * @demo demo/index.html
 */
class ButtonElement extends
  Vaadin.ElementMixin(
    Vaadin.ControlStateMixin(
      Vaadin.ThemableMixin(
        Polymer.GestureEventListeners(Polymer.Element)))) {

  static get is() {
    return 'vaadin-button';
  }
```

Figure 40. vaadin-button source code.

In Figure 40, the documentation in the comments shows four attributes: active, disabled, focus-ring, and focused. Sometimes, the component is configured in a special way that not all properties are exposed in the code. Hence it is important to read its designated documentation as well.

Types

Use the different types to prioritize the actions presented to the end user. Usually, you should only have one primary button in a view. Use tertiary buttons to decrease the priority of an action.

The `"tertiary-inline"` type is useful when you want to add a button as part of regular text content.

Primary Secondary Tertiary Tertiary Inline

```
<vaadin-button theme="primary">Primary</vaadin-button>
<vaadin-button>Secondary</vaadin-button>
<vaadin-button theme="tertiary">Tertiary</vaadin-button>
<vaadin-button theme="tertiary-inline">Tertiary Inline</vaadin-button>
```

Figure 41. Vaadin-button documentation on vaadin.com (Vaadin 2019a).

Figure 41 shows the documentation of the component on its main website, and it talks about the property *theme*, which users can combine different types, colors, sizes, and icons. For example, setting *theme="success primary large icon"* with the icon plus results in Figure 42.

```
<!-- Icon in place of label -->
<vaadin-button theme="success primary large icon" aria-label="Add new item">
  <iron-icon icon="lumo:plus"></iron-icon>
</vaadin-button>
```

Figure 42. Vaadin-button with custom styling by using attributes.

Knowledge gained from section 6.3.2, 6.3.3, and 6.3.4 helps to find more testing cases for this property test. Upon finishing checking all the information, a code snippet is made on CodePen to test the running of vaadin-button. The CDN from section 6.2 is used to import the web component.

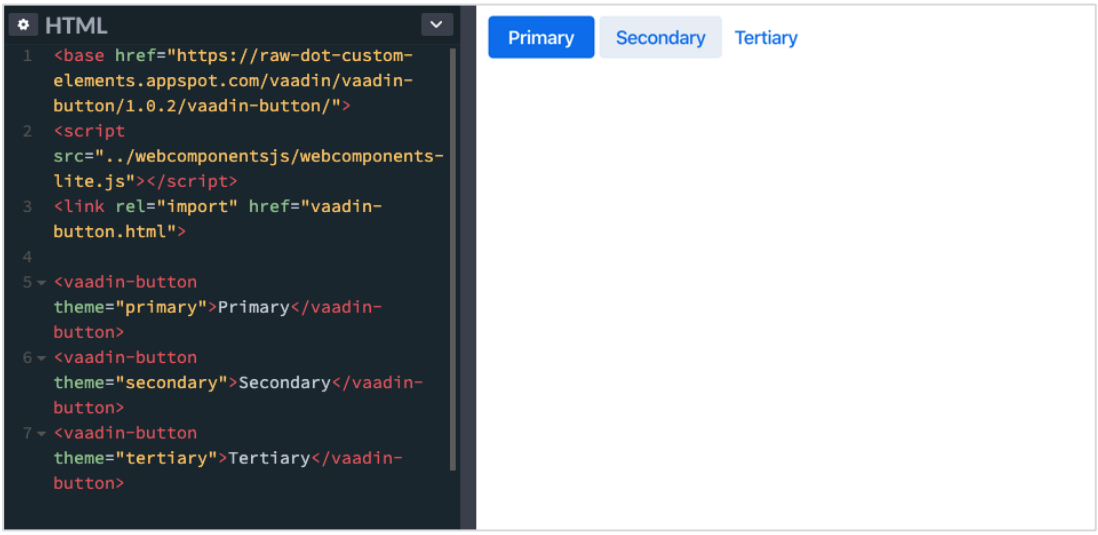


Figure 43. Vaadin-button demo on CodePen (Bui, B. 2018).

In Figure 43, the vaadin-button can be seen functioning normally in the Chrome browser. Then, different usage of the properties will be tested.

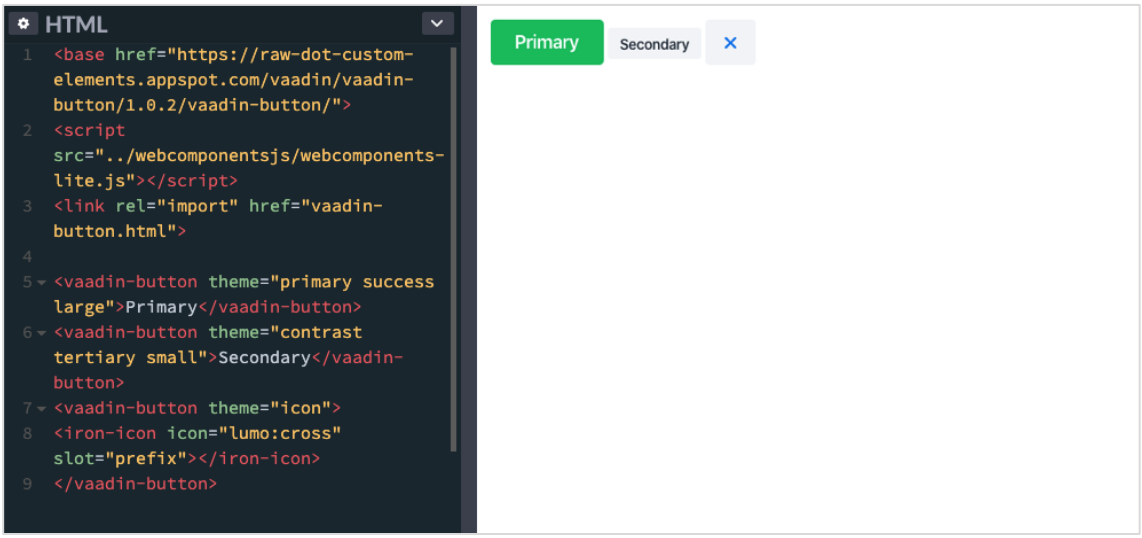


Figure 44. Vaadin-button properties test.

Testing the responsiveness of the component is the next task. Every modern web browser has Developer Tools, which include lots of designated features for a developer (MDN 2019f). One of those features is the Device Mode, which can emulate the viewport of different devices like mobile or tablet (Apple 2019, MDN 2019d, Basques 2019, Microsoft 2017). Device Mode is used to test the responsiveness of vaadin-button.

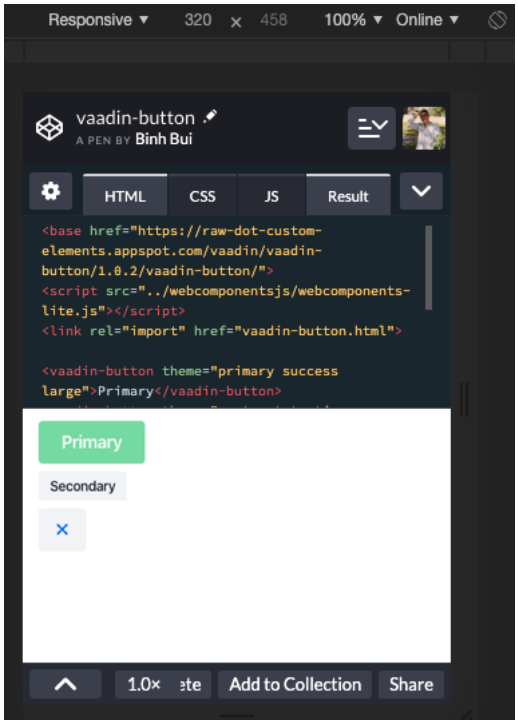


Figure 45. Vaadin-button responsiveness test.

Figure 45 signifies that vaadin-button is responsive under the mobile view. On many occasions, the component is responsive under one browser but not in another browser. Therefore, it is crucial to repeat the same test for all browsers.

When all the tests are executed on Chrome, the CodePen snippet will also be tested in other browsers. The conducted tests on other browsers follow the process as above, which is skimming the description, testing the properties, and checking the responsiveness.

This process of testing one component will be applied for all the rest of the Polymer 2 components.

7.6 Test result recording

All the results are inserted into the spreadsheet. First, the component gets a general feedback.

1.0.2: Fast loading and very easy to use, also responsive. You can combine different kind of theming by using just the theme property. It has good documentation and demo.

Figure 46. Example of a testing feedback.

Then, the initial of each browser is input if the component works on that browser. For example, CSOFIE means that it works on all browsers, while CO means that it only works on Chrome and Opera.

There are two status columns: *tested* and *include*. If the component has been tested, mark Y in *tested*. The column *include* states that if the component is going to be imported into Vaadin Directory, the input is either a Y or N.

While testing the component in CodePen, a screenshot of the visual is taken and uploaded to online storage. Then, the link to the screenshot is input. Also, the link of the CodePen snippet is input into the spreadsheet. Finally, a rating between 1 to 5 is given, and a category is chosen for the component, whether it belongs to UI, Data, Theming, or Miscellaneous section.

A	B	D	E	I	J	K	L	M	O	P	Q
repo	owner	github	wc	screens	samplec	supp	rati	comment	te	i	categ
vaadin-button	vaadin	https://githu	https:/	https://dl.dro	https://coder	CSOFIE	5	1.0.2: Fast loading and very easy to use, also responsive. You can combine different kind of theming by using just the theme property. It has good documentation and demo.	Y	Y	UI

Figure 47. Vaadin-button with all assessment information.

8 RESULT AND STATISTICS

8.1 Results

The whole testing process finished in 3 months, with 696 tested web components. This process produced a spreadsheet full of useful information and statistics. Additionally, a collection of web components demo on CodePen, and a collection of web components screenshots were obtained, as well.

8.2 Statistics

The advantage of storing all the data in a spreadsheet is that statistics can be gained easily from those data. Statistics are essential for looking at the big picture. Below is a collection of statistics that have been obtained from the testing process.

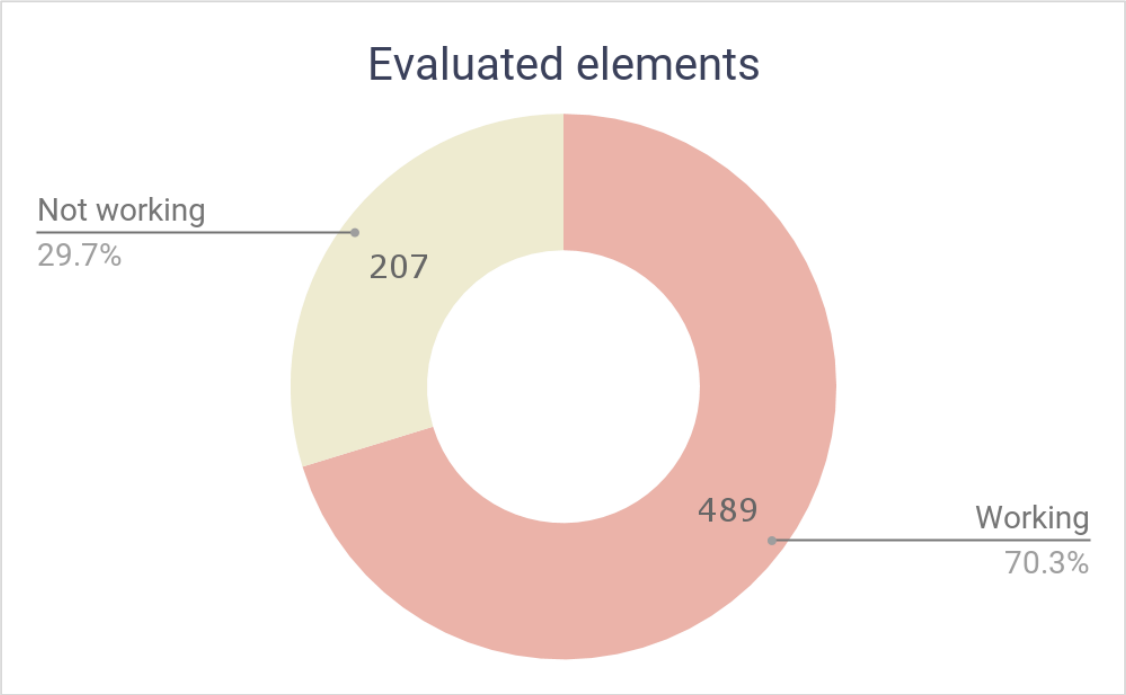


Figure 48. Number of functioning versus non-functioning components.

Out of 696 tested web components, 489 of them work, and 207 do not. The 207 components could practically work, but a lack of proper documentation and demo makes

it difficult to test it. As the time spent on testing each component is relatively short, there might be a margin of error here.

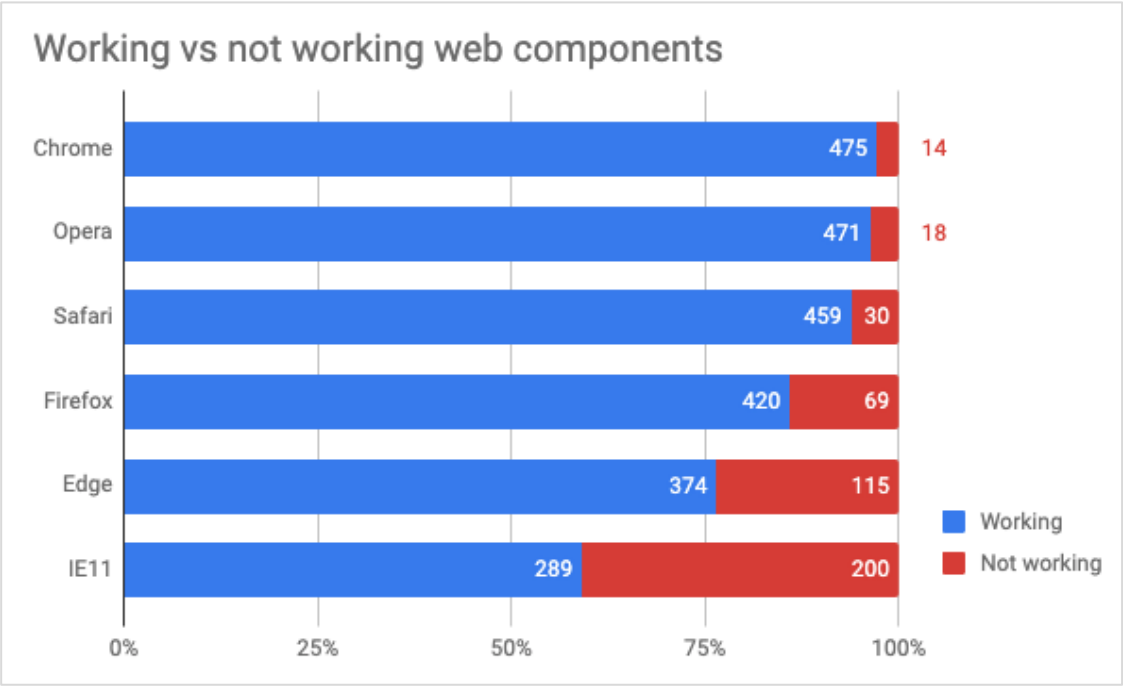


Figure 49. Number of functioning components in each browser.

As the most popular browser, Chrome aces at the top by supporting the highest number of web components. It can be easily predicted that Opera and Safari were taking up in the next places. As Opera and Safari are using the same web engine base as Chrome and Chrome is adopting the Web Components rather early, the resources it takes for Opera and Safari to enable the same standards will be less. Firefox gets 4th place, and undoubtedly, Edge and IE11 have the worst performance due to low support for the standards.

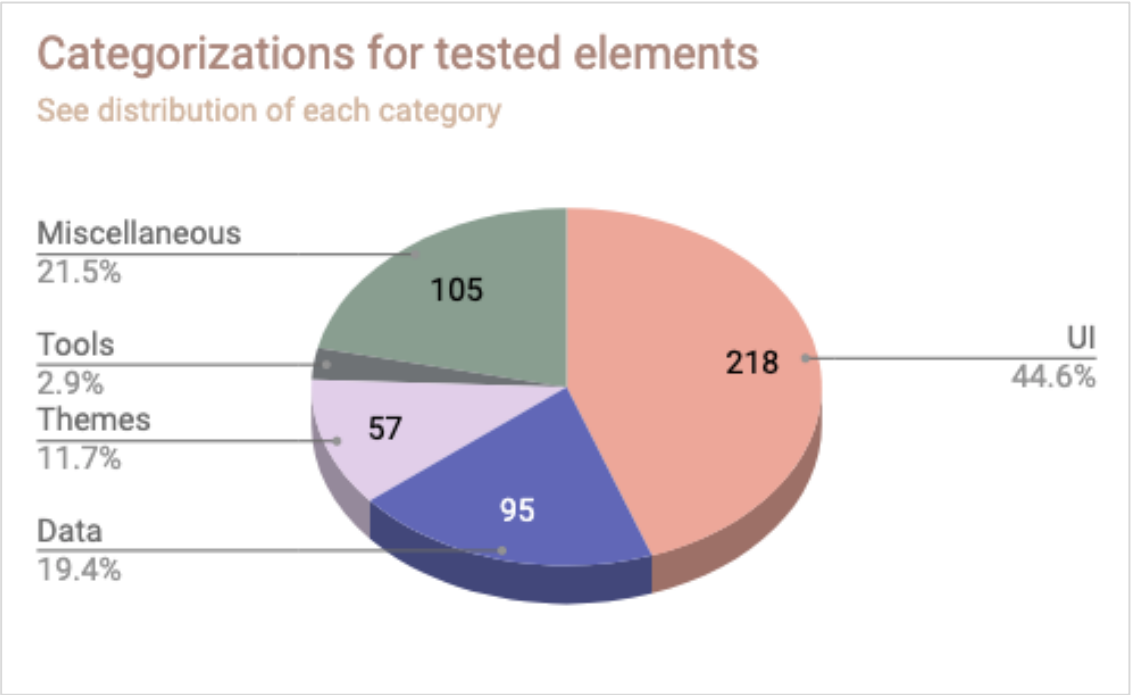


Figure 50. Distribution of functioning components' categories.

The UI section holds the largest number of web components, as it is relatively easy and less time-consuming to develop a UI component. It is worth to mention that UI components are the most demanded in web development. The Data section follows at the 2nd place since data manipulation is also highly sought in web development.

8.3 Importing web components into Vaadin Directory

All the 489 functioning web components were imported into Vaadin Directory with their corresponding category, rating, comment, library, screenshot, and demo link (Ekblad 2018a).

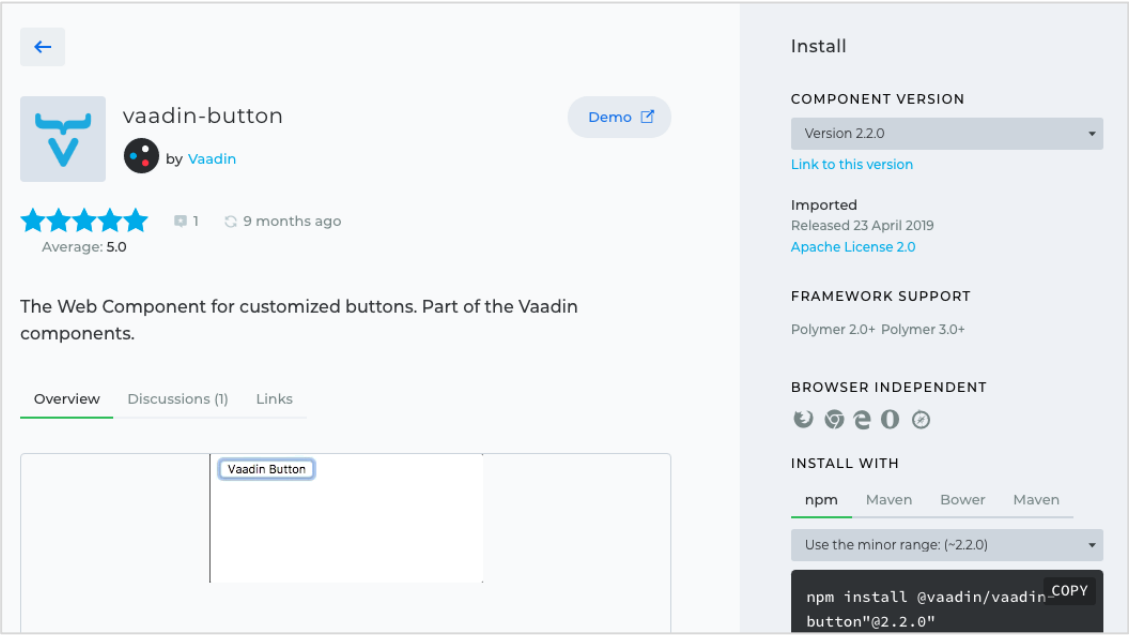


Figure 51. Vaadin-button in Vaadin Directory.

In Figure 51, the preview of vaadin-button in Vaadin Directory consists of all the obtained data from the testing process. On the left side, there is the rating, the Demo link, which uses the CodePen snippet, the Overview section, which includes the taken screenshots, the Discussions section, which includes the written feedback. On the right side, there is the web component library and browser support information.

A new filtering condition for Polymer web component is created so visitors can check only the web components section.

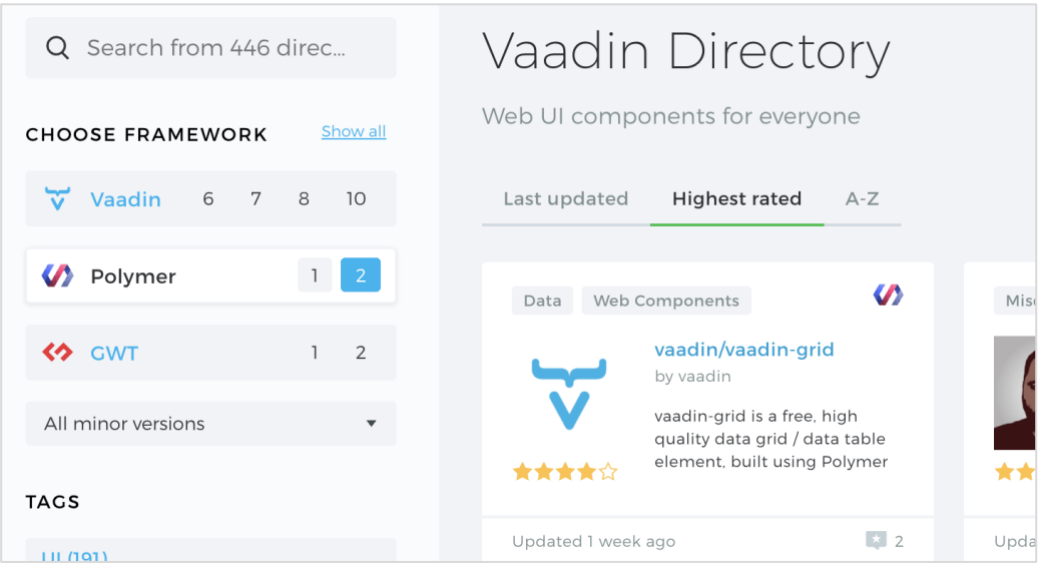


Figure 52. Polymer web components in Vaadin Directory (Ekblad 2018).

9 CONCLUSION

The goal was to get a general knowledge of Web Components standards, find the usage share of Web Components libraries, and test web components from webcomponents.org. The API feature from webcomponents.org was massively assisting in getting the usage share quickly. However, this thesis reached a domain that has never been touched before, which is to test hundreds of web components. As a result, a significant amount of time was spent on specifying the testing strategy and the implementation. In summary, the greatest challenge faced in this testing process was not about the testing complexity, but rather the ambiguity of the testing strategy.

Accordingly, there are plenty of possibilities where this testing process can improve, such as having a kit of automated tools or a better set of testing standards. Nevertheless, the knowledge gained from this research is massive both from the author's and Vaadin's perspective. In addition, this testing process obtained a great deal of useful data. Vaadin can utilize these data to assist the process of integrating the Web Components standards into the Vaadin framework. Also, Vaadin can positively apply the best practices from these data to develop a reliable set of components. The reliability includes functioning under all browsers, being responsiveness, and having proper documentation plus demo. The Vaadin Directory became populated with hundreds of hand-pick quality web components, along with plenty of handy information from the spreadsheet. Vaadin Directory is believed to help existing users with a lot more components to use and to encourage JavaScript web developers to evaluate the Vaadin framework.

While Web Components usage is ubiquitous these days, the learning of the Web Components standard and its journey illustrates how difficult it is for a proposed set of standards to be accepted and adopted by the browsers. Google has fought a long battle for catalyzing the adoption of Web Components, and the standard is expected to shape how websites are built in the future.

REFERENCES

ANGULAR, 2019-last update, Angular Elements Overview. Available: https://angular.io/guide/elements.

APPLE, 2019-last update, Web Development Tools. Available: https://developer.apple.com/safari/tools/.

BASQUES, K., 2019-last update, Simulate Mobile Devices with Device Mode in Chrome DevTools. Available: https://developers.google.com/web/tools/chrome-devtools/device-mode [Dec 3, 2019].

BUI, B., 2018-last update, vaadin-button demo. Available: https://codepen.io/binhbbbb/details/YxVvVV [Dec 3, 2019].

BUI, T., 2019-last update, Web components : concept and implementation. Available: http://www.theseus.fi/handle/10024/170793 [Nov 28, 2019].

CARL, -07-24T20:19:36+00:00, 2018-last update, Are Web Components Finally a Thing? Available: https://www.carlrippon.com/are-web-components-finally-a-thing/ [Nov 30, 2019].

CHROME, 2017-last update, Chrome Platform Status - HTML Imports. Available: https://www.chromestatus.com/features/5144752345317376.

CIMPANU, C., 2015-last update, Google Releases Polymer 1.0, a Library for the Modern Web Developer. Available: https://news.softpedia.com/news/Google-Realeses-Polymer-1-0-a-Library-for-the-Modern-Web-Developer-482689.shtml [Nov 28, 2019].

COLLABORNE, 2015-last update, iron-timer GitHub repository. Available: https://github.com/Collaborne/iron-timer.

EISENBERG, R., 2019-last update, Aurelia Release Notes - April 2019. Available: https://aurelia.io/blog/2019/04/02/aurelia-release-notes-april-2019/.

EKBLAD, S., 2018a-last update, Vaadin Directory now also lists Web Components. Available: https://vaadin.com/blog/vaadin-directory-now-lists-also-web-components [Nov 29, 2019].

EKBLAD, S., 2018b. *Vaadin Directory number of components monthly statistic.*

EVANS, A., 2019-last update, Web components: the secret ingredient helping power the web. Available: https://web.dev/web-components-io-2019/.

FRONTEERS, -12-09T19:36:39-05:00, 2011-last update, Alex Russell | Web Components and Model Driven Views | Fronteers 2011 [Homepage of Fronteers], [Online]. Available: https://vimeo.com/33430613 [Nov 28, 2019].

GORE, A., -05-21T00:00:00.000Z, 2018-last update, Create & Publish Web Components With Vue CLI 3. Available: https://vuejsdevelopers.com/2018/05/21/vue-js-web-component/ [Nov 29, 2019].

KESTEREN, A.V., 2014-last update, Mozilla and Web Components: Update – Mozilla Hacks - the Web developer blog. Available: https://hacks.mozilla.org/2014/12/mozilla-and-web-components [Nov 29, 2019].

LEHTINEN, J., -03-18 14:04:39 UTC, 2015-last update, Vaadin & Web Components [Homepage of Joonas Lehtinen], [Online]. Available: https://www.slideshare.net/joonaslehtinen/vaadin-web-components [Nov 30, 2019].

LEHTINEN, J., 2010a-last update, Vaadin Directory Demo [Homepage of Vaadin], [Online]. Available: https://www.youtube.com/watch?v=LSXCNL6Q1w0 [Nov 29, 2019].

LEHTINEN, J., 2010b-last update, Vaadin Directory Opens its Doors. Available: https://vaadin.com/blog [Nov 30, 2019].

MDN, 2019a-last update, The Document Base URL element. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base [Dec 2, 2019].

MDN, 2019b-last update, HTML attribute reference. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes [Nov 25, 2019].

MDN, 2019c-last update, HTML Imports. Available: https://developer.mozilla.org/en-US/docs/Web/Web_Components/HTML_Imports [Nov 28, 2019].

MDN, 2019d-last update, Responsive Design Mode. Available: https://developer.mozilla.org/en-US/docs/Tools/Responsive_Design_Mode [Dec 3, 2019].

MDN, 2019e-last update, Web Components. Available: https://developer.mozilla.org/en-US/docs/Web/Web_Components [Oct 26, 2019].

MDN, 2019f-last update, What are browser developer tools? Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_are_browser_developer_tools [Dec 3, 2019].

MICROSOFT, 2017-last update, DevTools - Emulation - Microsoft Edge Development. Available: https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide/emulation [Dec 3, 2019].

MICROSOFTEDGE, -07-15T17:00:06+00:00, 2015-last update, Microsoft Edge and Web Components. Available: https://blogs.windows.com/msedgedev/2015/07/15/microsoft-edge-and-web-components/ [Nov 29, 2019].

NIWA, R., 2019-last update, 2019 TPAC F2F. Available: https://github.com/w3c/webcomponents/issues/826.

O'SHAUGHNESSY, P., 2017-last update, Because Browser Diversity Is Good For The Web. Available: https://samsunginter.net/Because-Browser-Diversity-Is-Good-For-The-Web/ [Dec 2, 2019].

PANDAS, 2019-last update, DataFrame. Available: https://www.geeksforgeeks.org/python-pandas-dataframe/ [Nov 17, 2019].

PAUL, J., 2017-last update, Where is Java used in Real World? Available: https://javarevisited.blogspot.com/2014/12/where-does-java-used-in-real-world.html [Nov 29, 2019].

POLYMER, 2019-last update, Publish - webcomponents.org. Available: https://www.webcomponents.org/publish [Nov 29, 2019].

POLYMER, 2018a-last update, Browser support overview. Available: https://polymer-library.polymer-project.org/2.0/docs/browsers.

POLYMER, 2018b-last update, Custom element concepts. Available: https://polymer-library.polymer-project.org/2.0/docs/devguide/custom-elements [Nov 25, 2019].

POLYMER, 2018c-last update, Data Binding. Available: https://polymer-library.polymer-project.org/2.0/docs/devguide/data-binding [Nov 25, 2019].

POLYMER, 2018d-last update, Declare Properties. Available: https://polymer-library.polymer-project.org/2.0/docs/devguide/properties#attribute-reflection [Nov 25, 2019].

POLYMER, 2018e-last update, Document your elements. Available: https://polymer-library.polymer-project.org/2.0/docs/tools/documentation.

POLYMER, 2018f-last update, Observers and computed properties. Available: https://polymer-library.polymer-project.org/2.0/docs/devguide/observers#computed-properties [Nov 25, 2019].

POLYMER, 2018g-last update, Property change notification events. Available: https://polymer-library.polymer-project.org/2.0/docs/devguide/properties#notify.

POLYMER, 2018h-last update, What's new in 2.0 | Polymer. Available: https://polymer-library.polymer-project.org/2.0/docs/about_20.

POLYMER, 2018i-last update, Who's using Polymer? Available: https://github.com/Polymer/polymer/wiki/Who's-using-Polymer%3F.

PÖNTELIN, T., 2015-last update, The Future of Directory is here. Available: https://vaadin.com/blog/future-of-directory-is-here [Nov 30, 2019].

REVILL, L., -06-10T18:22:52.161Z, 2016-last update, Why web components are so important. Available: https://blog.revillweb.com/why-web-components-are-so-important-66ad0bd4807a [Nov 28, 2019].

SCHOOLEY, B., 2015-last update, Creating Web Components in Ember.js 2.0. Available: https://www.twilio.com/blog/2015/09/creating-web-components-in-emberjs.html [Nov 29, 2019].

SIMILARTECH, 2019-last update, Java VS NodeJs - Framework Technologies Market Share Comparison. Available: https://www.similartech.com/compare/java-vs-nodejs [Nov 29, 2019].

SOMESH, 2016-last update, Why, Where And Who Uses Polymer.JS? - Hashnode. Available: https://hashnode.com/post/why-where-and-who-uses-polymerjs-cimevacmv01x9tg53803u0rd8 [Nov 28, 2019].

STEGNER, B., 2017-last update, How to Use VirtualBox: User's Guide. Available: https://www.makeuseof.com/tag/how-to-use-virtualbox/ [Dec 2, 2019].

VAADIN, 2019a-last update, HTML Examples | Button | Components. Available: https://vaadin.com/components/vaadin-button/html-examples/button-theme-variants-demos [Dec 2, 2019].

VAADIN, 2019b-last update, Vaadin - The best Java framework for Progressive Web Apps. Available: https://vaadin.com/ [Oct 29, 2019].

WANG, H., 2018-last update, Vaadin 10: Bring Web Components to Java Realm [Homepage of Liferay], [Online]. Available: https://www.youtube.com/watch?v=9YFL_MwFKFI [Nov 29, 2019].

WEBCOMPONENTS, 2018-last update, Introduction - webcomponents.org. Available: https://www.webcomponents.org/introduction [Nov 28, 2019].

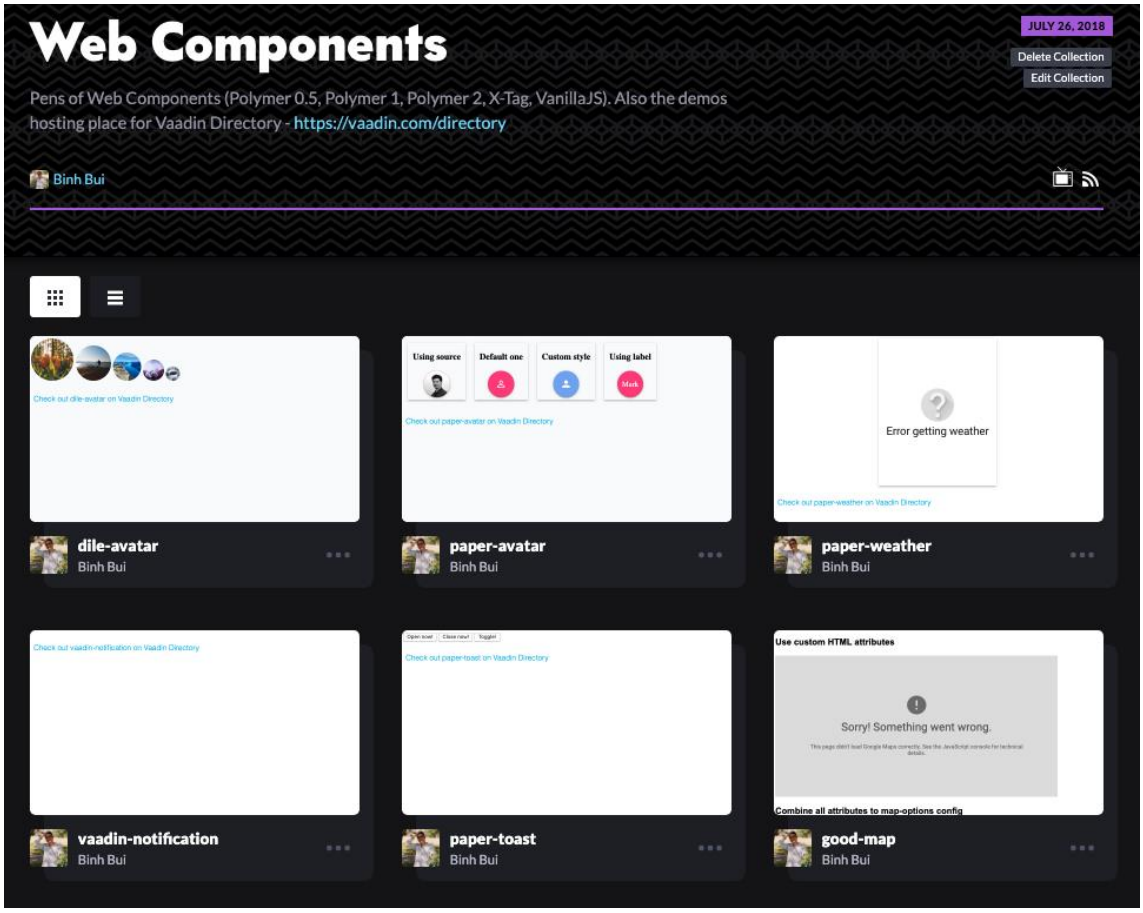
WEBCOMPONENTS, 2017-last update, webcomponents.org API. Available: https://github.com/webcomponents/webcomponents.org/blob/master/API.md [Nov 25, 2019].

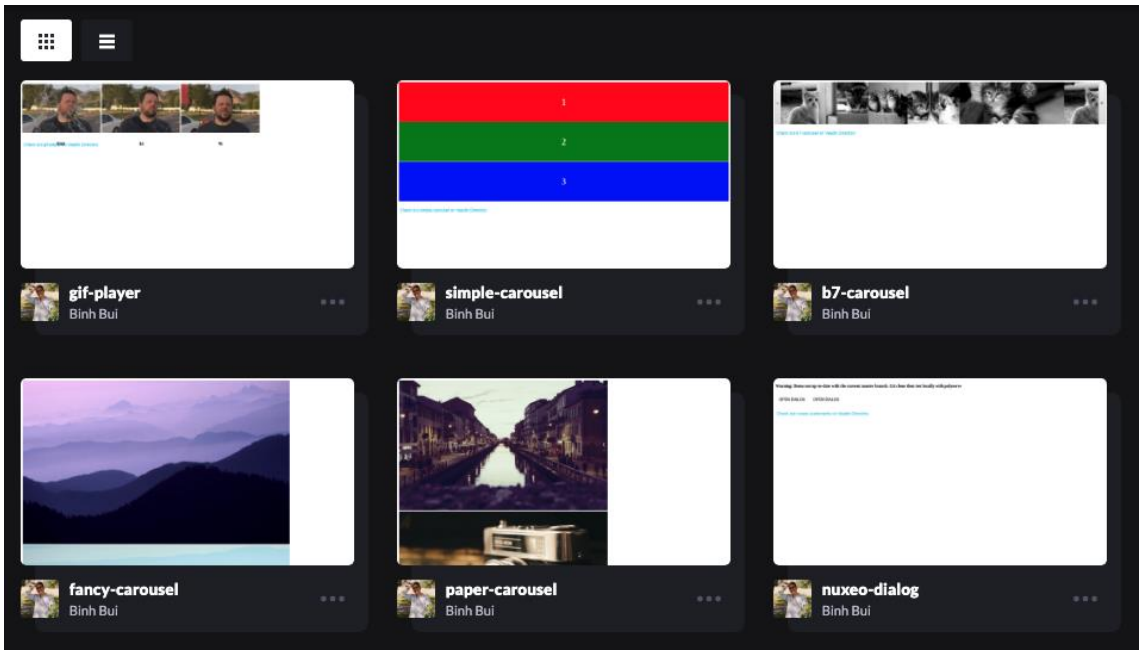
WEBCOMPONENTS, 2016-last update, webcomponents.org repository on GitHub. Available: https://github.com/webcomponents/webcomponents.org.

WEBCOMPONENTS.ORG, 2019-last update, Status of modern browsers supporting Web Components standards. Available: https://www.webcomponents.org/ [Nov 28, 2019].

WYTREBOWICZ, T., -04-02T18:48+00:00, 2018-last update, W3C Web Components meeting. Available: https://starcounter.io/w3c-web-components-meeting-%f0%9f%97%bc%f0%9f%87%af%f0%9f%87%b5/ [Nov 29, 2019].

Collection of web components demo on CodePen





Web Components

JULY 26, 2018

Delete Collection

Edit Collection

Pens of Web Components (Polymer 0.5, Polymer 1, Polymer 2, X-Tag, VanillaJS). Also the demos hosting place for Vaadin Directory - https://vaadin.com/directory

Binh Bui

Title	Created	Last Updated		Stats	
dile-avatar	December 4, 2018	December 4, 2018	0	80	0
paper-avatar	December 4, 2018	December 4, 2018	0	86	0
paper-weather	November 19, 2018	November 21, 2018	0	188	0
vaadin-notification	November 6, 2018	November 6, 2018	0	109	0
paper-toast	November 5, 2018	November 7, 2018	0	95	0
good-map	September 25, 2018	September 25, 2018	0	228	0
paper-swatch-picker	August 10, 2017	September 25, 2018	0	222	0
mist-password	September 25, 2018	September 25, 2018	0	103	0
paper-masked-input	September 25, 2018	September 25, 2018	0	107	0
gold-password-input	September 25, 2018	September 25, 2018	0	104	0
voice-player	September 25, 2018	September 25, 2018	0	119	0
voice-recognition	September 25, 2018	September 25, 2018	0	133	0
gif-player	September 25, 2018	September 25, 2018	0	120	0
simple-carousel	September 25, 2018	September 25, 2018	0	116	0
b7-carousel	September 25, 2018	September 25, 2018	0	122	0
fancy-carousel	September 25, 2018	September 25, 2018	0	123	0
paper-carousel	September 25, 2018	September 25, 2018	0	227	0
nuxeo-dialog	September 24, 2018	September 24, 2018	0	68	0
m2-dialog	September 24, 2018	September 24, 2018	0	83	0
dialog-el	September 24, 2018	September 24, 2018	0	81	0
paper-alert-dialog	May 14, 2018	September 24, 2018	0	94	0
emoji-rain	September 24, 2018	September 24, 2018	0	108	0
emo-ji	September 24, 2018	September 24, 2018	0	67	0
emoji-reaction-selector	September 24, 2018	September 24, 2018	0	73	0
vaadin-dropdown-menu	September 21, 2018	September 21, 2018	0	297	0
tsante-dropdown	September 21, 2018	September 26, 2018	0	191	0
smart-accordion	September 3, 2018	September 3, 2018	0	482	0
paper-autocomplete	August 27, 2018	August 27, 2018	0	135	0
pokemon-selector	August 27, 2018	August 27, 2018	0	157	0
usa-state-list	August 26, 2018	August 27, 2018	0	184	0
paper-bottom-sheet	August 24, 2018	August 24, 2018	0	29	0
iron-timer	August 24, 2018	August 24, 2018	0	47	0
wired-menu-item	August 23, 2018	August 23, 2018	0	29	0
progress-bar	August 20, 2018	August 20, 2018	0	127	0
paper-pulsating-progress	August 18, 2018	August 21, 2018	0	102	0
tk-loading-backdrop	August 18, 2018	August 18, 2018	0	112	0
animated-content-placeholder	August 18, 2018	August 18, 2018	0	224	0
vaadin-text-field	August 15, 2018	August 15, 2018	0	1	0
alump-dbar	August 14, 2017	July 26, 2018	0	70	0
bowser-polymer	November 29, 2017	November 29, 2017	0	179	0

Blogs written by the author regarding the testing process

Web Components statistics in 2017

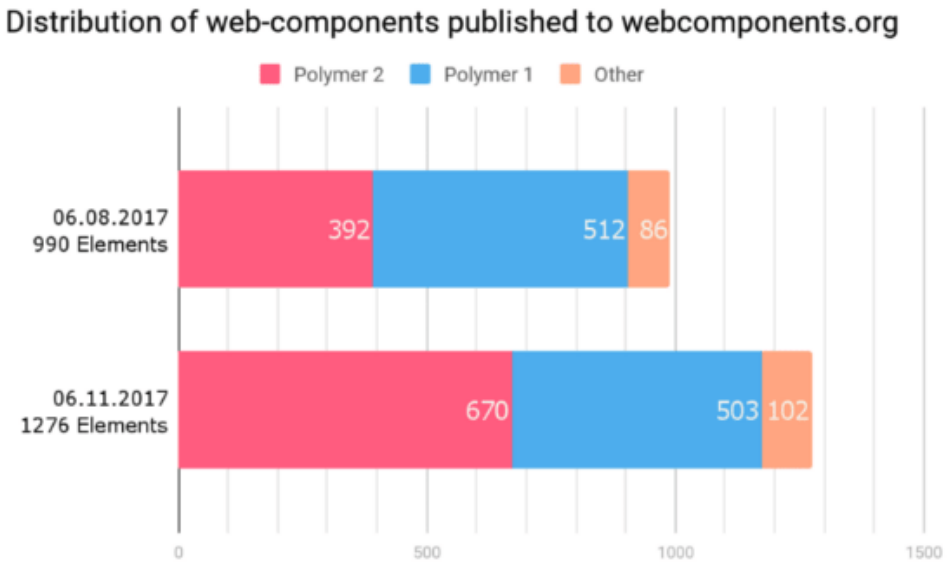
Binh Bui Jul 30 '18 · 2 min read

#javascript #webcomponents #statistics #beginners

This article is the last story in the series "Testing 696 Web Components". The statistics in this article covers many topics: the number of components published on webcomponents.org, distribution of components based on its libraries, working and non-working components, and some more interesting stats.

Note: These statistics were yielded roughly a year ago. Some of them might become irrelevant in the current day.

300 elements in 3 months. That's an impressive number.



The story of a man who has tested 696 Web Components

Binh Bui

Jul 19, 2018 · 4 min read

> *"Do you know what Web Components is? How would you test hundreds of them?" — Sami Ekblad, my supervisor in our interview*

That was the beginning of my journey to explore the world of web components. I, myself, was a 22-year-old IT student with no professional experience. And to make it even more interesting, that interview was the first time I heard the term "Web Components."

Starting from zero

Things were rough in the beginning. No one in our company has done this task before, or anyone else publicly. There were no testing guidelines, no formats, no nothing. This was the scopes list I come up with:

- Functionality: Is it working?
- Design: How does it look?
- Compatibility: Does it work on all platforms?

I begin by manually picking some from ~~webcomponents.org~~ and start testing them. Luckily, they feature JSFiddle demo (~~like this~~) for previewing components. The downside is that they are available only if the author provides the sample code. Nonetheless, those inline demos are the little sunshine! 🌞

Top 5 obstacles I faced in testing 696 web components

Binh Bui
Jul 26, 2018 · 4 min read

Following my last autobiography, I promised to post a story concerning the toughest barriers I faced during testing web components.

The fact is that, when the number of components reaches hundreds, there will be thousands of those obstacles you meet along the way. In this story, I will point out the 5 most common barriers.

No reliable content delivery network (CDN)

In short, CDN offers a fast, efficient and secure delivery of content to websites. To test remotely on sites like CodePen, JSFidde or JSBin, a reliable CDN is heavily needed.

Otherwise, you need to test it locally, which includes downloading the source code, installing the dependencies and serving the demo on the website (I say it's triple, if not more, the amount of work).