FACULTEIT ECONOMIE EN BEDRIJFSKUNDE

# QUALITY EVALUATION OF MODERN FRONT-END WEB DEVELOPMENT FRAMEWORKS.

## A PRACTICIONERS FRIENDLY GUIDE

Wordcount: 13853

Studentnumber : 01504685

Promotor: Prof. Dr. Els Clarysse

Master's Dissertation submitted to obtain the degree of: Master of Science in Business Administration – management and IT

Academic year: 2019-2020

UNIVERSITEIT GENT

# Table of Contents

# Preface

## List Abbreviations and definitions

| | |
|---|---|
| AJAX | Asynchronous JavaScript and XML |
| API | Application Programming Interface |
| CLI | Command Line Interface |
| CSR | Client-Side Rendering |
| CSS | Cascading Style Sheet |
| DOM | Document Object Model |
| GQM | Goal, Question, Metric |
| HTML | Hyper Text Mark-up Language |
| HTTP | Hyper Text Transfer Protocol |
| IEC | International Electrotechnical Commission |
| ISO | International Organization for Standardization |
| JS | JavaScript |
| JSF | JavaScript Framework |
| JSX | JavaScript XML |
| KPI | Key Performance Indicator |
| MPA | Multi Page Application |
| MVC | Model-View-Controller |
| MVVC | Model–View–Viewmodel |
| NPM | Node Package Manager |
| SEO | Search Engine Optimization |
| SPA | Single Page Application |
| SSR | Server-Side Rendering |
| TAM | Technology Adoption Model |
| UI | User Interface |
| UTAUT | Unied Theory of Acceptance and Use of Technology |
| VDOM | Virtual Document Object Model |

## List Figures and tables

# Content

## 1. Introduction

The JavaScript programming language evolved on different aspects since its introduction in 1995 as a simple client-side scripting language. Today it is the most prominently used programming language to date according to the Stack Overflow Developer Survey of 2019 (figure 2). The community surrounding JavaScript is also on par with its popularity, in GitHub's 2019 The state of the Octoverse report, JavaScript is ranks first in terms of amount of pull and push requests of code on GitHub. Furthermore, the JavaScript software ecosystem also exceeds the size of other languages, NPM is the world's largest ecosystem of open source libraries in the world and hosts over 1 million open source packages (DeBill, 2020). This provides developers with a lot of resources, reusable functionalities, and code to apply in their respective projects.

The rise of the language can be attributed to several factors. The primary one being, that it is the lingua franca for web applications. High portability is another advantage, it can run on any device that runs a browser be it a TV, mobile phone, or tablet.  The Language has even moved beyond the browser. It is now possible to develop native mobile applications in JavaScript thanks to Frameworks which bridge JavaScript code into native device components (Heitkotter, Majchrzak, Ruland, & Weber, 2013). Adding to this versatility JavaScript can also be used as a server-side language thanks to the introduction of the Node.js framework, which is a JavaScript runtime built on Google's V8 runtime engine, it allows for development of Fullstack applications entirely written in JavaScript (nodejs, 2020).

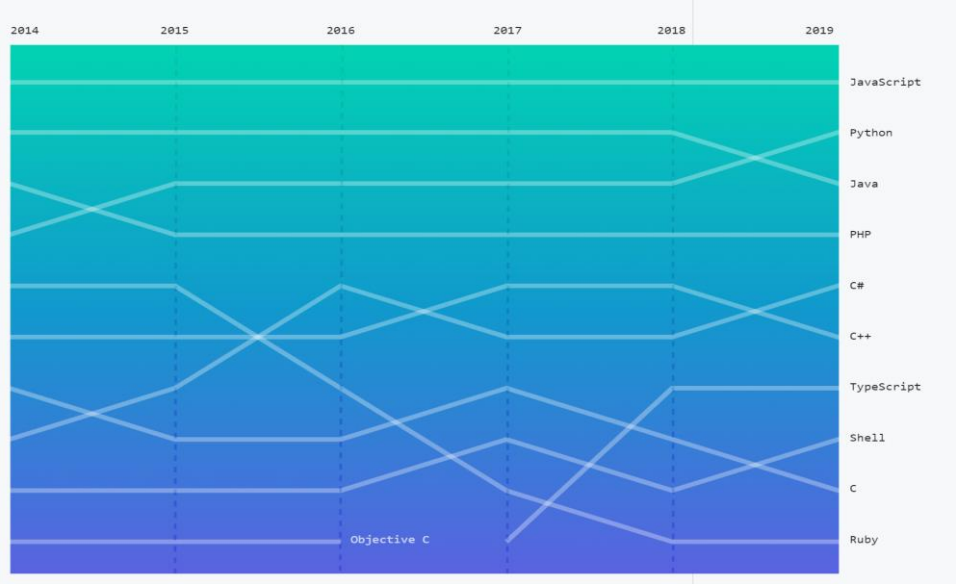


*Figure 1: Top Language over time (GitHub Octoverse, 2019)*

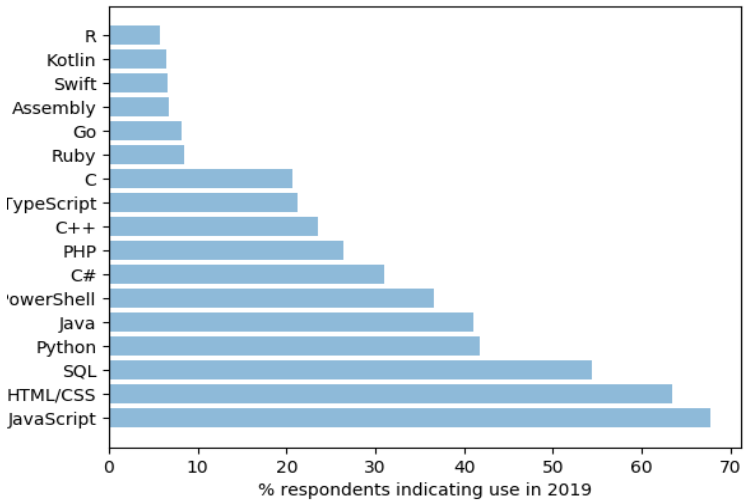



*Figure 2: Respondents indicating using programming language (Stack Overflow Developer Survey, 2019)*

The result of this popularity is a wide variety of libraries and frameworks aimed at addressing JavaScript Developers needs and requirements. Companies and developers thus must make multiple decisions amongst a large amount of options, concerning the technologies to use when building web applications (Graziotin & Abhramsson, 2018). In full-stack applications both server and client-side architecture are taken into consideration and given the fast evolving and dynamic nature of the industry this is a difficult task. More concretely, a wide selection of client-side JavaScript frameworks have emerged which is why some now call it "a jungle of JavaScript Frameworks" (Graziotin & Abrahamson, 2013). Amongst the wide variety of choices, the most popular front-end JavaScript frameworks of 2019 were Vue.js, React and Angular. In the survey the state of JavaScript 20 000 JS Developers responded to a survey addressing their views and preferences concerning the JS ecosystem. The result shows that Vue, React and Angular remain the most widely used frameworks (figure 3).

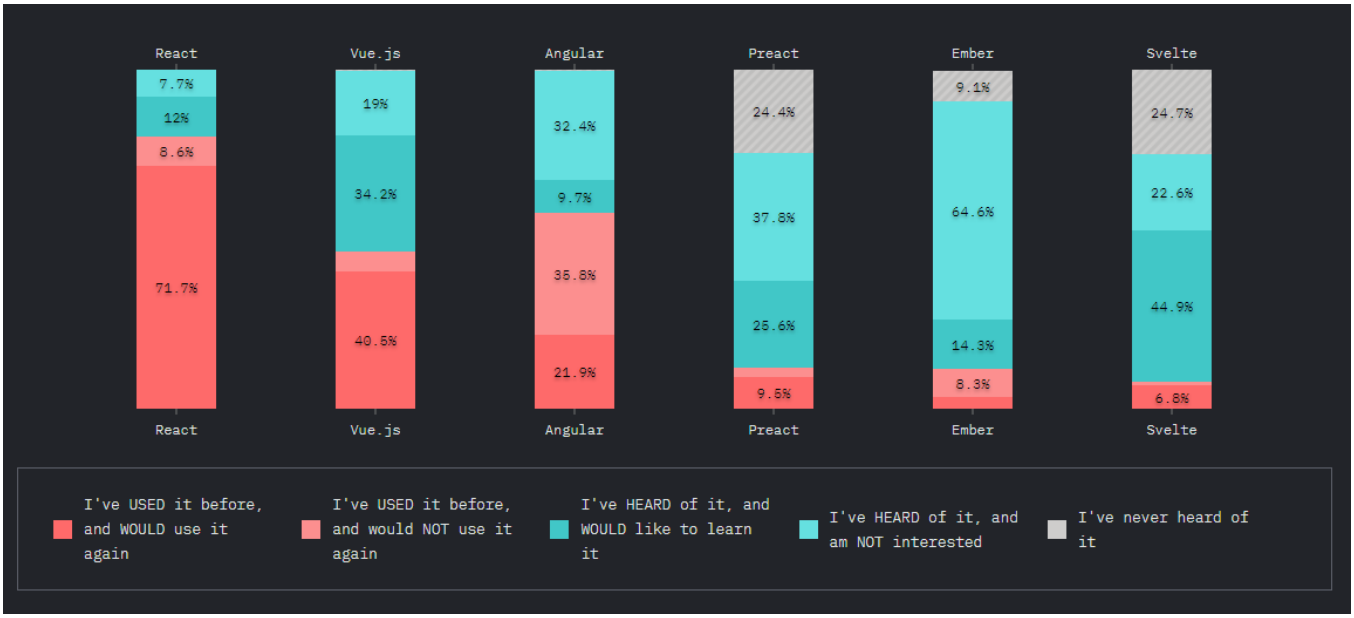



*Figure 3: Overall popularity of front-end frameworks (State of JavaScript, 2019)*

Because of their popularity and adoption rate these frameworks were chosen to be examined in greater detail. Even though they do not fully operate in the same way they when developing an application, a choice will often be made between them. They all embrace component-based web development and handle reactivity within applications which is why they are grouped together (Mraz, 2019).

## 2. Motivation

In 2013 Graziotin & Abrahamson made a call for action to develop a practitioner's friendly evaluation framework which can guide the selection of a front-end JSF. According to them software metrics are often used as a benchmark when a framework is evaluated while at the same time practitioners indicate that this is not their primary selection criteria. Five years later they followed up their research by placing the adoption factors for JavaScript Frameworks within the UTAUT model (figure 4) (Graziotin, Abhramsson, & Pano, 2018). The model developed serves as a baseline for other quantitative and qualitative studies concerning JSF adoption or as a foundation for benchmarking different frameworks. Considering this insight this paper expands upon their work but with a focus on three front-end frameworks, Vue.js, React.js and Angular. The motivation for this selection is both popularity and usage amongst practitioners.

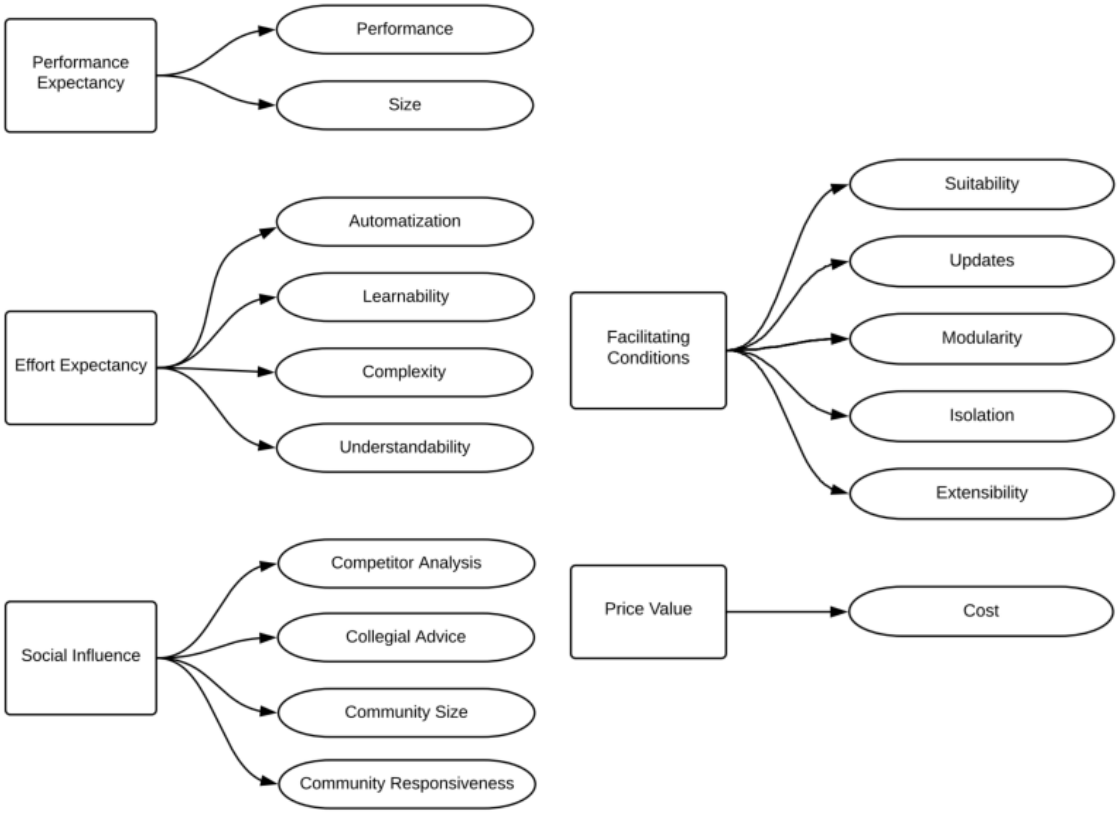



*Figure 4: Factors leading to the adoption of a JavaScript Framework* (Graziotin, Abhramsson, & Pano, 2018)

To guide this research the ISO SQUARE standards process will be followed this choice is motivated by the fact that context dependency is considered an important factor for technology related evaluation frameworks. Within figure 4 social influence and facilitating conditions are not intrinsic traits of the technology itself but rather part of the context in which the technology is used. A similar approach is taken in several similar papers in which software is evaluated. Christoph & Majchrzak (2019) for example conducted research comparing JSF for mobile development wherein they conclude that ranking the frameworks is not a viable approach as it would lead to the proverbial "comparison of apples and oranges" within different use cases. Heitteköter et al. (2013) also consider context as a crucial factor when evaluating JSF's for cross-platform development. They constructed a framework where practitioners can assign their own weight to each of the criteria in function of their specific requirements and needs. In short, the selection of software is a subjective process dependent on the needs, preferences and complexity of the project and its developers (Teixeira, Xambre, Alvelos, Filipe , & Ramos, 2015). Because similar papers have chosen this approach it was also seen as a factor that has to be considered when selecting a front-end JSF.

In conclusion, Finding the right architecture for the development of an application is vital given that a wrong decision will have an impact on the entire product (Losavio, Chirinos, Matteo, Lévy, & Ramdane-Cherif, 2004). The choice of a framework or libraries is part of that choice and thus a crucial step in the development process, which requires a careful evaluation. Because several benefits arise when the right solution is implemented. Some of the added benefits can be, a faster development process, more structured code, reusability of components and concurrent updates (Pekhanova, 2009). This is even more true in the case of open source software compared to commercial software where reliability support and performance are assured by a relationship with a supplier open source software offers no warranties. The result of this is that open-source software has to be studied beforehand to ensure that it is viable for production (Teixeira, Xambre, Alvelos, Filipe , & Ramos, 2015). A context dependent evaluation framework could help address some of these concerns.

## 3. Research question

In response to this papers motivation two research questions are put forward:

**RQ1:** How can a front-end JSF be defined?

- What is a front-end framework?
- Which features and characteristics define Vue.js, Angular and React.js?

**RQ2:** Depending on the context of use, which characteristics of front-end frameworks should be taken into consideration to find the most suitable one?

- Which technical and non-technical characteristics are of interest?
- How can these characteristics be reflected in practice?

In what follows the front-end web-development landscape is characterized and an evaluation framework is established which, providing practitioners with a tool which factors in technical and non-technical aspects of the frameworks to help determine which one is best suited for their specific use cases or project. To develop such a tool the ISO 25000 software quality model development process is followed (ISO, 2019). This methodology offers a general model that can identify quality factors of a software product. The model must however be adapted to the specific software products which are evaluated, in this case front-end JSF.

## 4. Structure

The first area of this paper answers RQ1, a literature study is conducted which serves two objectives. First, defining and comparing the JSF that are subject of the study. Second, serve as a starting point for the research conducted to address RQ2. After the literature study an initial evaluation framework and interview questions will be established. The literature study first defines what a quality model is and how it can be applied to front-

end frameworks. Afterwards, a brief history of frontend development is given from the viewpoint of frontend frameworks and in conclusion the current most popular frameworks are further examined on their most defining characteristics. The second area of this paper follows the process described within the ISO SQUARE standards for which the methodology is elicited in part six. This includes the methodology used during the stakeholder's requirements definition and analysis process (figure 5) and the software quality evaluation process for which a sample application will be developed. Within chapter seven the study itself is described which includes the stakeholder requirement analysis via interviews and the evaluation process by creating an application with each of the frameworks. Afterward part eight discusses the limitations and scope, suggestions and results of the study. To conclude with chapter nine where the study is summarized.

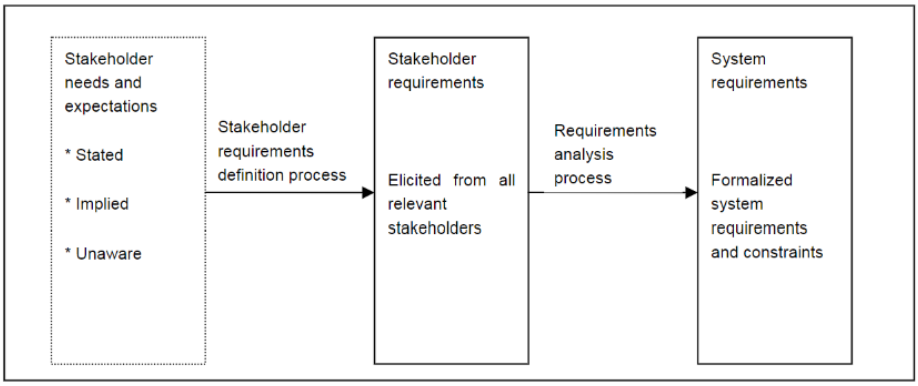



*Figure 5: Stakeholders requirements process* (ISO, 2019)

## 5. Literature Study

### *5.1 Quality and evaluation models*

In the ISO 250N0 standards Software Quality is defined as:

> *"The capability of a software product to satisfy stated and implied needs when used under specific conditions" (ISO, 2019).*

"Under specific conditions" signifies quality dependency on the context of evaluation and viewpoints of the people making the evaluation. Quality is thus not considered absolute but rather quality balances on a scale of degree (ISO, 2019). Quality is also a multidimensional concept and which is why it is important to know which aspects are important for evaluation (Curcio, Malucelli, Reinehr, & Antônio, 2016). According to the ISO SQUARE standards software product quality can be measured on three areas. First, internal attributes meaning

the static properties of the product itself. Next, external attributes which measures behavior of software when executed and finally, quality in use.

Within the standards a general set of quality characteristics and sub-characteristics related to software products are presented which represent these three areas. How much each of the characteristics should be taken into consideration will depend on the context of use. Context of use being is defined as the users, tasks, equipment, and the physical and social environments in which a product is used (ISO, 2013). The SQUARE series itself provides guidance to do adapt the characteristics, operationalize them and use the result for a specific use case. The standard includes five divisions each specifying a respective part of the process mentioned before (table 1).

| *name* | *description* |
|---|---|
| Quality management division | General reference which defines common models, terms and definitions for the other divisions. Guidance for managing requirements specification and evaluation. |
| Quality model division | Provides quality models for system and product quality and quality in use but also for data quality which is not relevant in our case. |
| Quality Measurement Division | Presents internal and external measures of software quality and quality in use measures. |
| Quality Requirements Division | Helps with the specification of quality requirements for a specific context. |
| Quality Evaluation Division | Provides requirements, recommendations and guidelines for practical implementation of quality evaluation. |

*Table 1: Divisions within SQUARE standards (own creation)*

The standard offers two processes to follow (figure 6). The software quality requirements specification process and the software quality evaluation process, which is underlyingly supported by the software quality measurement process. In short, the standard serves as a starting point of eliciting software product requirements, how to measure those criteria and how to evaluate them.

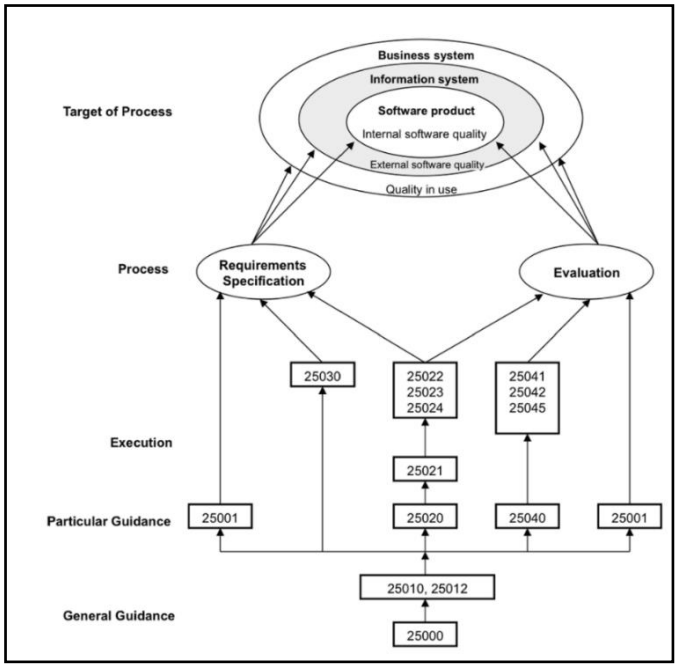



*Figure 6: SQUARE general reference model* (ISO, 2019).

5.1.1 Requirements definition and analysis process

ISO/IEC 25010 serves as a starting point for making a quality model. it includes two generic models for quality evaluation which are adjustable to fit a context of use (figure 7).

- **The quality in use model:** five characteristics which will be defined by the outcome of using the software in a specific context.
- **the product quality model:** This includes all characteristics and sub-characteristics that can be considered when evaluating a software products quality.

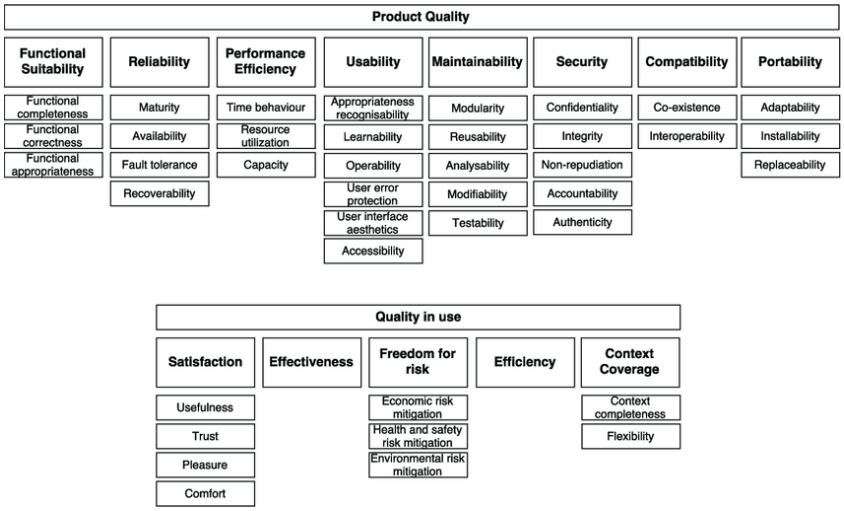



*Figure 7: external and internal software product quality (sub-)characteristics*
*And quality in use (sub-)characteristics* (ISO, 2019)

The two models are linked as analysis of in use requirements will result in derived functional and quality requirements for the product. Taking up all (sub-) characteristics is not advised, instead the model must be tailored to the product evaluated in order to take up the most prominent factors (ISO, 2019).

The discovered needs and expectations of stakeholders are the result of the requirements elicitation and definition process (figure 5). A requirements definition process aims to state the requirements for a product to fulfill the needs of stakeholders and users in a defined environment. As a result, the context of use and required characteristics are defined. The requirement analysis process aims to transform the requirements into a technical view which offers a product that lives up to the requirements. Both the internal software requirements for the frameworks (the white box – how they operate), the external (how they perform when executing) and quality in use are considered. After definition and specification of these characteristics they must be operationalized in the form of quality properties for which measurements can be made.

- **Internal software quality measure**: measures degree in which set of static attributes of software product satisfy stated and implied need (static attributes include those relating to software architecture, structure, and its components)
- **External software quality measure**: measures degree to which a software product enables the behavior of a system to satisfy stated and implied needs for the system including the software to be used under specific conditions.

- **Quality in use measurements**: measures degree to which product or system can be used by specified users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific a specific context of use

The attributes found in the ISO standards (figure 7) have a big overlap JSF adoption factors identified by Abrahamson et al. shown in (figure 4). Table 2 illustrates this idea; within it the characteristics and adoption factors cover grouped based on similarity. For adoption factors however value was left out given that the frameworks we evaluate are open source.

| *Main JSF adoption factor* | *Sub JSF adoption factor* | *Main characteristic ISO* | *Sub Characteristic ISO* |
|---|---|---|---|
| **Performance expectancy** | Performance | Perfomance efficiency | Time behaviour |
| | Size | Perfomance efficiency | Resource utilization |
| **Effort expectancy** | learnability | Usability | learnability |
| | understandability | Usability | operability / appropriatness recognizability |
| | complexity | functional suitability | functional correctness |
| ***social influence*** | competitor analysis | satisfaction | trust |
| | collegial advice | satisfaction | usefulness |
| | community size | context coverage, satisfaction, reliability | context completeness, trust, maturity |
| | community responsiveness | satisfaction | trust |
| **facilitating conditions** | suitability | funtional suitability | functional completeness, correctness and appropriatness |
| | updates | portability | adaptability |
| | modularity | maintainability / context coverage | modularity, modifiability / flexibility |
| | isolation | functional suitability | functional completeness |
| | extensibility | portability | adaptability |

*Table 2: mapping of JSF adoption factors to ISO SQUARE quality models (own creation)*

5.1.2 Quality evaluation with quality models

Each of the quality properties should be measurable by applying a measurement method (figure 10). Which makes characteristics and sub-characteristics quantifiable. ISO/IEC 25021 offers quality measure elements that can be used to construct software quality measures. Quality measure elements are the base and derived measures used to create measures of software product quality characteristics. Quality measure elements may measure a static representation of the software, the behavior of the software, or the effects of the software when it is used.

- **Internal measures** characterize software product quality based upon static representations of the software

- **external measures** characterize software product quality based upon the behavior of the computer-based system including the software, and
- **quality in use measures** characterize software product quality based upon the effects of using the software in a specific context of use.

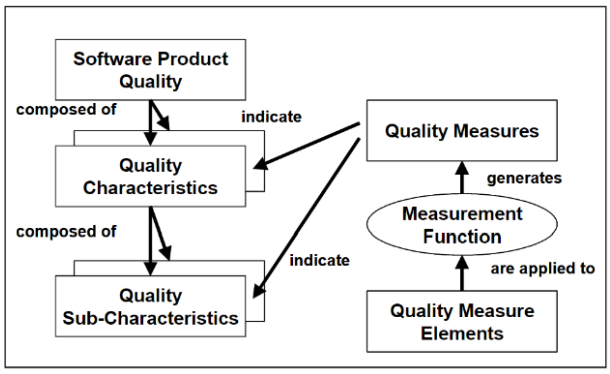



*Figure 8: Software product quality measurement reference model* (ISO, 2013)

Software metrics have historically primarily focused on performance benchmarks. But increasingly today has become less relevant for web development as hardware power increases (Pekhanova, 2009). Rather several external factors such as social context, perception and context are becoming more important to practitioners as the field is evolving, (Graziotin, Abhramsson, & Pano, 2018).

5.1.3 Scope of quality models

A system is a combination of interacting elements organized to achieve one or more stated purposes (ISO, 2013) .Three factors in quality models will influence the quality in large. These three factors are, the business system it is used in (figure 9), the type of stakeholders evaluating and the period in the lifecycle (figure 12). The standard recognizes that software in general is part of a bigger system in which it is used and the requirements for the system are closely connected to the requirements for the software which is why they cannot be viewed in isolation. Systems have multiple stakeholders with their own specific needs and requirements. As an example, developers may want to use a new technology while the project manager prefers a more stable technology which already has an established reputation. The timing of the evaluation can also have an impact as requirements might change depending on the stages of development. The system, stakeholder and timing for this paper are the final application to develop within a company's IT department, the developers working with said front-end technologies and the development stage. these aspects are further clarified in the methodology section.

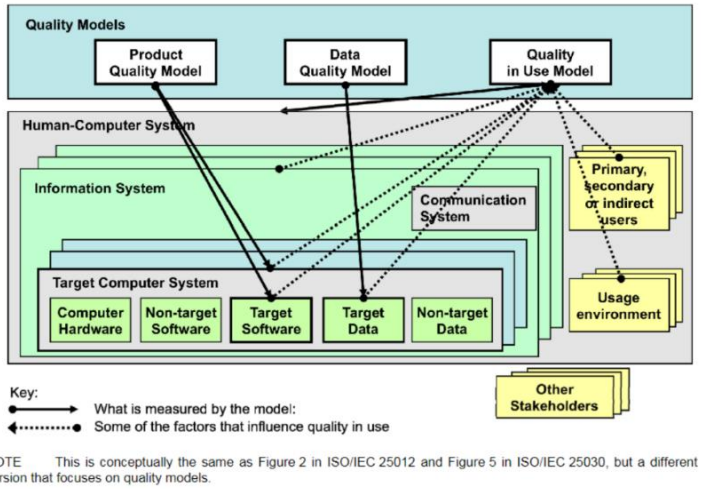



*Figure 9: Targets of quality models*

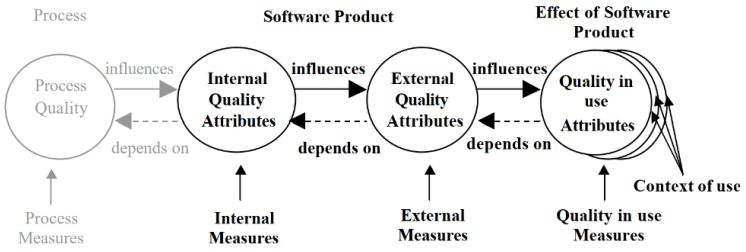



*Figure 10: Quality in the lifecycle*

## *5.2 Front-End web-development frameworks*

### 5.2.1 Background and evolution of front-end development

A traditional web application consists of two basic building blocks, a back-end or server-side and a front-end or client-side. The client side can be defined as the code executed on the device of the user while server side can be defined as executed outside of the user's device which then sends an output to the user (Amjad, Hanaan, & Moiz, 2017). Technologies used for front-end development are rapidly changing with new libraries, tools and frameworks being released and updated in quick fashion (Aquinno & Gandee, 2016). At the core however most of the front-end development consist of the technologies HTML, JavaScript, and CSS (figure 11). On top of the core technologies a plethora of libraries and frameworks were made of which the ones discussed in this paper are currently very popular (Taivalsaari, Mikkonen, Cesare , & Systä, 2018).

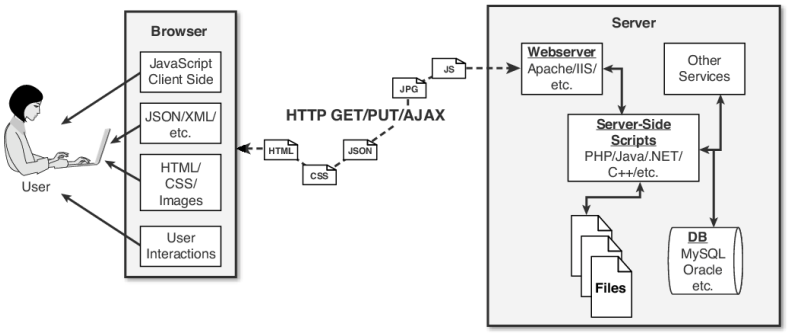



*Figure 11: Diagram of the components of a basic website/web application.* (Dayley B. , 2014)

An overview of the history of front-end development shows why this is the case. In the early stages of the world-wide web much of the scripting for creating websites took place on the server side because the Web was originally very document-oriented, and it was at that time unforeseeable that it would become a massive application-oriented, content distribution environment (Taivalsaari, Mikkonen, Systä, & Pautasso, 2018). In recent years however interactivity and functionality of web-based applications has increased to the point that web applications have come to behave like their desktop counterparts (Hanchett & Listwon, 2018). This means an increase in user interaction and data exchanges. Web applications are now increasingly asynchronous and interactive which supported the rising interest in front-end development as it responsible for handling interactions whether they come from the user or the system and determining how the user interface, and stored data should react to it (Bibeault & Kat, 2008).

A big driver in the evolution towards asynchronous behavior was the introduction of AJAX, and the improvement of JavaScript engines in the browser (MacCaw, 2011). Ajax is not a single technology but an approach in web development in which XMLHTTP-request objects are implemented to make requests asynchronously after a page has been loaded as illustrated in figure 12 (Galli, 2019). XML within the name can, be misleading given that the AJAX approach can be used to send non-XML data such as JSON or text files as well.

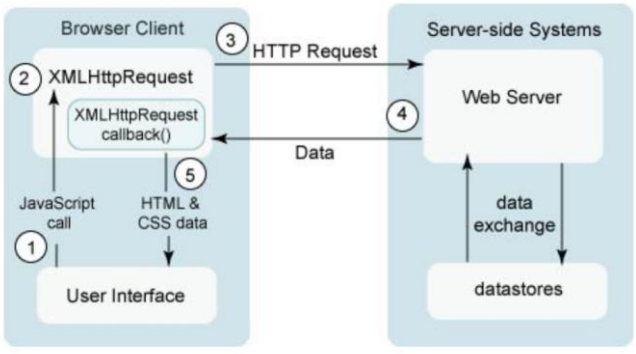



*Figure 12: AJAX requests* (Aquinno & Gandee, 2016).

Because developers where obliged to keep up with this increase in complexity the volumes of unmaintainable client-side code also increased. In response to this evolution frameworks and libraries emerged which made client-side scripting easier (Hanchett & Listwon, 2018). jQuery is one of the earliest examples, it is a JavaScript library that provides an easy-to-use API designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax requests (jQuery, 2020). But thanks, of the evolution of the JavaScript language and browsers a lot of these initial functionalities became available by other means. For example the Dom selector API and fetch API for AJAX became a standard part of the browser as they were integrated into the World Wide Web Consortium specs and are now supported by almost all browsers (W3C, 2015) . jQuery is therefore although it is extremely widespread seldom chosen to develop a new application.

Instead new frameworks like React, Vue and Angular are now the standards because they provide significant benefits one example being, their declarative approach (Brown, 2019). Declarative programming is a programming style where the application is structured in a way that code should describe what should happen instead of how it should happen which is an imperative approach (Banks & Porcello, 2017). To illustrate this idea, consider figure 14 and 15 where with the jQuery solution you are required to explicitly create your DOM elements one by one via a loop while in Vue for providing a template describing data structure, logic and how it connects to the DOM is enough. In general, React, Angular and Vue excel at structuring the dataflow throughout an application allowing a focus on data instead of the DOM.

```
<body>
  <div id="output"></div>
</body>
<script>
  var shoppingList = ["Apples", "Bananas", "Milk"];

  $.each(shoppingList, function (index, value) {
    $("<div>" + value + "</div>")
      .click(function () {
        $(this).toggleClass("highlighted");
      })
      .appendTo($("#loop"));
  });
</script>
```

```
<body>
  <div id="output">
    <div
    v-for="item in shoppingList"
    v-on:click="item.selected= !item.selected"
    v-bind:class="{highlighted: item.selected}">{{ item.name }}</div>
  </div>
</body>
<script>
  new Vue({
  el: '#output',
  data: {
    shoppingList: [
      { name: 'Apples', selected: false },
      { name: 'Bananas', selected: false },
      { name: 'Milk', selected: false },
    ],
  },
})
</script>
```

*Figure 13: jQuery vs Vue.js solution example (own creation)*

As mentioned, ut supra Frontend development has evolved to use different libraries and or frameworks. the variety and functionalities of all these libraries and frameworks has thus over the years only increased. From full-fledged, opinionated MVVC frameworks like Angular to the light weight approach of Vue which promotes a progressive integration into projects (Taivalsaari, Mikkonen, Cesare , & Systä, 2018) (Hanchett & Listwon, 2018). Because of this wide variety a clear definition of functionalities and scope of the JSFs in this study is

needed. And although scope and functionalities differ, a comparison is possible given that a choice amongst these frameworks is often made when a client-side of a web application is developed given that they tackle a lot of the same general problems.

5.2.2 libraries and frameworks

Although they are commonly referred to as front-end JavaScript frameworks React.js describes itself as a library, Vue.js as a framework and Angular calls itself a platform. To avoid confusion terminology the meaning of these descriptions should be further clarified.

A framework is an implementation of a group of design principles and patterns which enable code reuse by abstracting away the structure and mechanisms of an application domain (Schmidt & Bushmann , 2003). In practice for front-end frameworks this means a developer can quickly scaffold a product line front-end architecture with a pre-set collection of boilerplate code, a directory structure and frequently used design principles. A Framework will thus define an application's structure in a fundamental way. A library on the other hand is considered a toolbox for developers out of which they can extract functionalities that they wish to apply in their application, which facilitates an easier combination with other libraries or functionalities (Nitze, 2014).

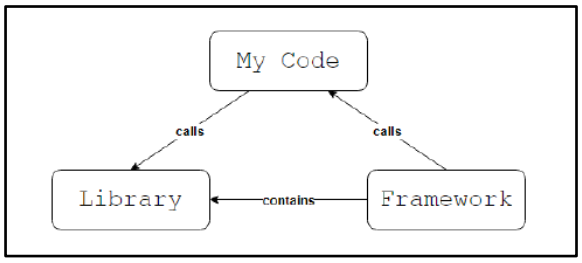



*Figure 14: Library vs framework philosophy* (Rastogi, 2019)

Out of the three frameworks React is considered the most flexible and liberal of the three, its core features include DOM manipulation, a component-based architecture and component level state management (ReactJS, 2020). All other functionalities are developed by the community which provides more freedom of choice in terms of the way the application is build (Wohlgethan, 2018). Vue.js describes itself as a progressive framework, with at its core a library only focusing on an applications UI or view (Vue.js, 2020) This means that React and Vue at their core almost completely provide the same functionalities. The difference here is however in the fact that Vue.js does not only include a view library but also offers additional optional libraries for state management or routing. Because the framework is progressive however developers are not obligated to use them but rather are given the choice if and when they want to integrate them. The fact that additional libraries

are included for the framework itself makes Vue.js more opinionated then React.js. Angular calls itself a platform because it is more holistic in comparison to a framework, it includes tooling and support that go beyond simply providing a code library to build an application (Wilken, 2018). Angular is the most opinionated of the three and when adopted it will dictate a large part of how the application is written and which patterns should be used, the fact that TypeScript has to be used for example is already a big decision which is taken out of the hands of developers.

In this paper Vue.js, React and Angular will remain to be referred to as front-end JavaScript Frameworks, because developers will often choose between these technologies when developing client-side of an application and given that is how they are commonly referred to when compared amongst each other. When considering React.js, Angular and Vue.js, the exact determination of vocabulary is not the primary focus of this paper instead it is meant to given an overview in terms of actual functionalities they provide and non-technical characteristics that can make one or the other more suitable for a given context.

5.2.3 frontend frameworks

React.js is currently most popular front-end development framework (figure 3), Facebook open-sourced in 2013 and maintains and updates the library.  Its goal is to help with the creation of user interfaces (ReactJS, 2020). This means the focus is set on component-based user interfaces and DOM manipulation. Google released the Angular framework in 2016 and currently maintains and updates it. It is a complete rewrite and successor of AngularJS which was released in 2010. Angular is considered the most feature rich framework of the three. The framework includes functionalities and solutions for most common front-end tasks and has an opinionated approach to it. Unlike the previous frameworks Vue.js was not created by a big tech company, instead the framework was released in 2013 by Ewan You, a former Google Engineer. Because of its growth in popularity however it has become a relevant player that is maintained by a core team and adopted by a variety of companies . Like with React.js the focus is on UI. Vue.js also markets itself as a progressive framework implying that users can implement Vue only on certain areas of an application and then incrementally implement extra functionalities depending on their needs.

| Functionalities | React | Angular | Vue |
|---|---|---|---|
| ***Syntax*** | JSX | Standard Typescript | Single file close to HTML, JS, and CSS |
| ***Impact on application architecture*** | weak | strong | medium |
| ***UI architecture*** | Component based | Component based | Component based |
| ***UI / DOM manipulation*** | Virtual DOM | real DOM | Virtual DOM |
| ***State Management*** <br> **** each is extensible with other state management libraries*** | Component level <br><br> High-level management via Context and Reducer api | Component level <br><br> High-level management via services | Within components <br><br> Core library Vuex |
| ***Rendering Solutions*** | Standard CSR SPA <br> - optional SSR (next.js most popular implementation) <br> - static site generation (Gatsby most popular implementation) | Standard CSR SPA <br> - optional SSR (Angular Universal) <br> - no static site generation | Standard CSR SPA <br> - optional SSR(nuxt.js most popular implementation) <br> - static site generation (nuxt.js most popular implementation) |
| ***Non-view related modules*** | Community libraries | Includes HTTP Client, routing module and form handling and validation | Core library Vue router |

*Table 3: Comparison table of frameworks (own creation)*

## ***Impact on Application architecture :***

All three of the frameworks shared parts architecture and functionality with one another (figure 15).

- A component-based approach to UI development, which ensures code reusability and encapsulation.
- UI and application state are kept in sync automatically.
- State can be managed within components.
- The standard solution for rendering SPAs with these frameworks is client-side rendering, although there are options available to implement different rendering approaches.

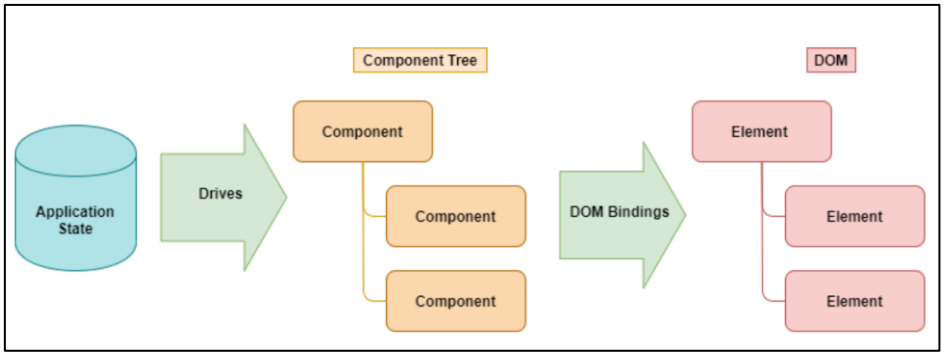



*Figure 15: Common Approach React, Angular and Vue* (Tyson, 2019)

Application architecture overall will be influenced by the framework used, one will be more opinionated and thus have a bigger impact than another. Angular for one is strongly opinionated while React is the most liberal of the three. Their degree of influence can be seen in how much support there is for external JavaScript libraries. In Angular external libraries are almost never required to be imported and usually also more difficult to implement while React offers an opposite approach Vue falls in the middle where they offer official Vue solutions for some functionalities but don't require a developer to use them. When viewed from the perspective of an MVC architecture Vue and React will focus mainly on Views while Angular aims to cover the Model and Controller as well.

***Syntax :***

React.js is by default used with the JSX syntax which stand for JavaScript and XML. JSX offers developers the capability to write full JavaScript code logic inside HTML mark-up. Unlike within Angular and Vue.js this offers developers the capability to write full JavaScript code within HTML. The syntax is just a feature to ease development however, when the code gets compiled it gets converted into JS which uses React.createElement() to creates a JS object that React reads when constructing the DOM (Banks & Porcello, 2017).

Angular was designed to write reactive SPA's with HTML and TypeScript, the framework consists of core and optional libraries which are all written in TypeScript. Angular's syntax differentiates itself by requiring development in TypeScript, which defines itself within the official documentation as a typed superset of JavaScript that compiles to vanilla JS (TypeScript, 2020). Although React and Vue also have the option for development with TypeScript this is not a default as is the case with Angular. One of the members of Angular's Core team summarizes the motivation for choosing TypeScript.

> *"TypeScript takes 95% of the usefulness of a good statically typed language and brings it to the JavaScript ecosystem. You still feel like you write ES6: you keep using the same standard library, same third-party libraries, same idioms, and many of the same tools (e.g., Chrome dev tools). It gives you a lot without forcing you out of the JavaScript ecosystem." (Savkin, 2016).*

Angular maintains a strong separation of concerns in the way files are structured, for each component styling logic and templating will be split in different files. React and Vue on the other hand choose for a separation along functional lines by grouping component into one component file based on its logical purpose.

In comparison to Angular and React.js it aims to stay as close to vanilla JS, CSS, and HTML as possible. To accomplish this goal Vue.js is most often used by writing .vue files. These are single files which include CSS styling, JS logic and HTML templating. The goal of this approach is to offer a solution that is approachable as possible for people only familiar with these core technologies while still offering the power a framework like React.js or Angular offers (Vue.js, 2020).

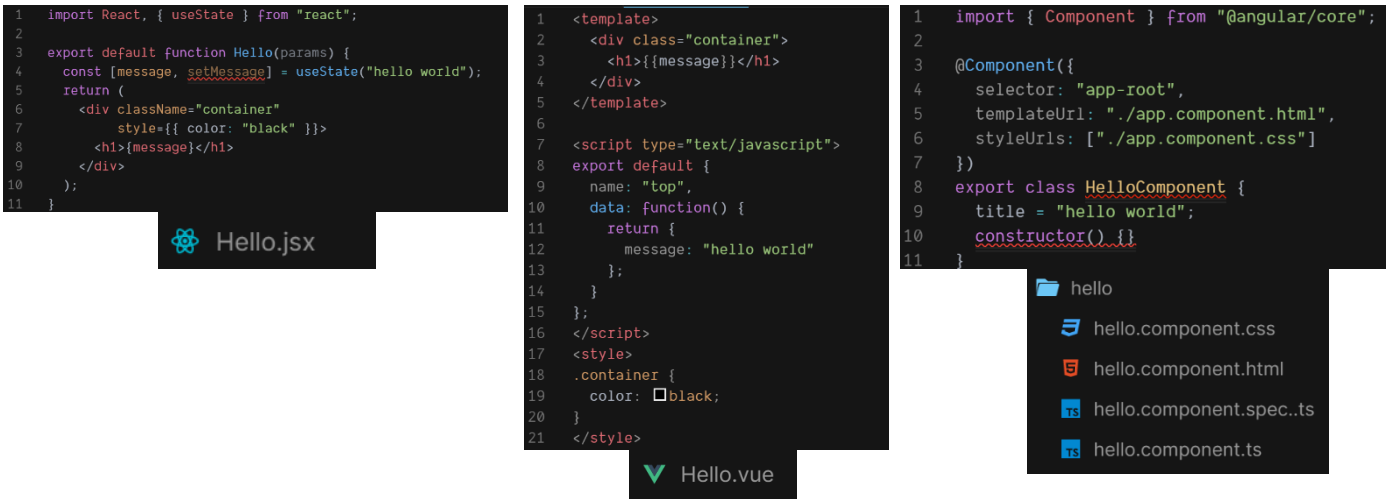



*Figure 16: Comparison of framework syntax (own creation)*

***Component based UI :***

In all three of the frameworks an application can be interpreted as a composed set of components. Each of the frameworks defines one root component from which a tree of nested components can be drawn. Components are therefore seen as the main building blocks of an application. One of the most important design principles for a component-based architecture is separation of concerns. This means encapsulating on a component level which makes software much more adaptable for change, maintainable and reusable (Mraz, 2019). With components logical parts are extracted out of code and separated into smaller logical parts which can be used throughout an application (Hanchett & Listwon, 2018). Because every UI consists of parts it makes sense to split an applications UI into reusable parts (Banks & Porcello, 2017). As in figure 17 where each part of a UI is split into a component, it is often logical to split up parts of your application based upon the single responsibility principle. Although the idea of component base web development was already introduced in 2011 by Alex Russell by introducing web-components to make the DOM more extensible. It was React.js in 2013 who first popularized this concept and because of its widespread success web components have now also emerged as a technology part of HTML5 specs, which means a component-based architecture is now easy to implement even without a framework.

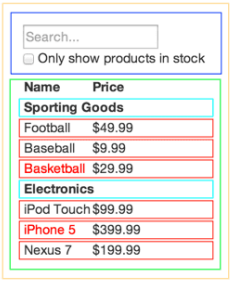



*Figure 17: component-based UI* (ReactJS, 2020)

## ***UI /DOM manipulation :***

Rendering in means converting data which describes the state of the UI into DOM objects which a browser uses to produce a visual UI. An essential part of the three front-end frameworks is to project application state within the UI. A framework will take data within JavaScript and project it into the DOM tree. The DOM refers to an API for HTML or XML documents, it defines how the documents are structured, accessed, and manipulated. Because of the DOM JavaScript developers can manipulate the document's content, structure, and style (Robie, 2020). To make the interaction with document easier the DOM will represent the document hierarchically as represented in the tree structure on figure 18 which illustrates that within the DOM every HTML tag is seen as an object each with their own properties and methods.

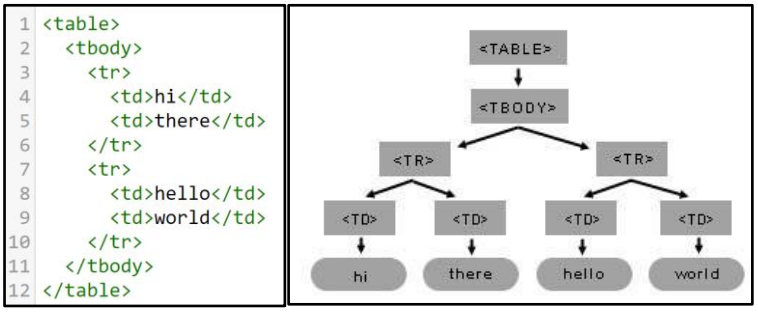



*Figure 18: Example of generated object tree* (W3C DOM standard, 2020)

DOM manipulations are straightforward when application data stays consistent. web pages however are often dynamic instead of static in nature, meaning that data model will change frequently in reaction to user- or other external events to which the view must adapt. Although re-rendering or updating of the DOM is possible with vanilla JavaScript the process can be complicated and very time consuming (Banks & Porcello, 2017). On top of this DOM changes trigger browser reflows which impacts performance in a negative way if performed excessively (Fink & Flatow, 2014).

Therefore, frameworks handle efficiently updating the view based on application state. Although each of the frameworks have a different way of doing this the goal remains the same, free up developers from tracking UI changes individually and instead program views declaratively. This offers a better solution because describing what a UI should look at any moment (declarative approach) is easier than describing how to change it (imperative approach) (Banks & Porcello, 2017) (ReactJS, 2020). To illustrate this concept, figure 19 show an example where UI changes based on information retrieved via an HTTP get request. In the vanilla JavaScript example, the steps involved in updating and rendering the UI are more verbose while when using a framework like Vue it is enough to just change the state of the application and let the framework handle the manipulation of the DOM and consequently render the update view areas.

```
const list = document.getElementById("list");
(async function update() {
  const response = await axios.get(
    "https://financialmodelingprep.com/api/v3/majors-indexes"
  );
  const indexes = response.majorIndexesList;
  indexes.forEach((index) => {
    const nested_list = document.createElement("ul");
    nested_list.className = "nested_list";
    nested_list.innerHTML = `
      <li class="list_item">Stock Index: ${index.indexName}</li>
      <li class="list_item">Index Price: ${index.price}</li>
      <li class="list_item">Index Change: ${index.changes}</li>
    `;
    list.appendChild(nested_list);
  });
})();
```

```
<body>
  <div id="app">
    <ul id="list"></ul>
  </div>
</body>
```

```
var app = new Vue({

    el: "#app",
    data: {
     indexes: [],
     response: false
    },

    beforeCreate: function() {
     axios
      .get("https://financialmodelingprep.com/api/v3/majors-indexes")
      .then(res => {
        this.response = true;
        this.indexes = res.data.majorIndexesList;
      })
      .catch(err => console.error(err));
    }
});
```

```
<body>
  <div id="app">
    <ul id="list" v-if="response" v-for="index in indexes">
      <ul class="nested_list">
        <li class="list_item">Stock Index: {{index.indexName}}</li>
        <li class="list_item">Index Price: {{index.price}}</li>
        <li class="list_item">Index Change: {{index.changes}}</li>
      </ul>
    </ul>
  </div>
</body>
```

*Figure 19: Updating views vanilla JS and Vue (own creation)*

To achieve view and state synchronization frameworks follow a two-step process of change detection and re-rendering. The process of Change detection is different for each of the frameworks, Angular's approach is based on using dirty model checking while React and Vue create a virtual DOM to ensure synchronization (figure 20).

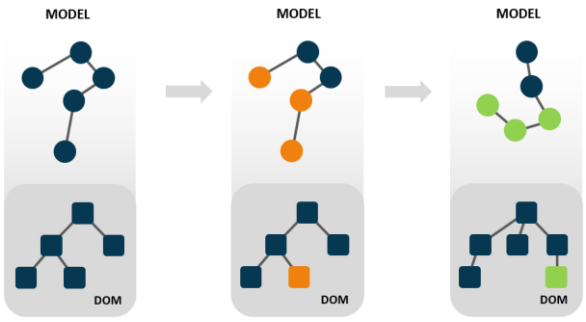



*Figure 20: Change detection and rendering* ( Koretskyi, 2018).

within Angular a change detector class is attached to each of its components during compilation. A library called zone.js enables Angular to listen for asynchronous tasks such as Event callbacks, network calls or timers. These scenarios trigger the detect changes function for detector classes attached to components. This function will then compare new bound model values to old ones which is what is called dirty checking. Afterwards the DOM nodes for which bound values have changed will update as required (Angular, 2020).

A virtual DOM, is a lightweight JavaScript object representation copying the real DOM. When an app gets rendered for the first time the virtual DOM is first created and then the structure gets created in the real DOM. Consequently, on each state change the entire virtual DOM will be re-created but because this is just a lightweight JavaScript object recreating it is computationally in-expensive. After a change there are then two different virtual DOMS on which an algorithm is applied to detect changes between them. Only those changes are then applied to the real DOM. This approach ensures a minimal update of the real DOM which would otherwise be computationally heavy process (Vue.js, 2020).

The approaches implemented to handle rendering and updates will have the biggest influence on runtime performance (after loading), which determines if a user sees an application as fast and responsive (Heitkotter, Majchrzak, Ruland, & Weber, 2013). table 4 shows however that differences in runtime performance are on average negligible for each framework.

| Name | angular-v5.0.0-keyed | react-v16.1.0-keyed | vue-v2.5.3-keyed |
|---|---|---|---|
| create rows<br>Duration for creating 1000 rows after the page loaded. | 185.7 ±7.8 (1.1) | 187.6 ±4.3 (1.1) | 169.2 ±3.6 (1.0) |
| replace all rows<br>Duration for updating all 1000 rows of the table (with 5 warmup iterations). | 179.3 ±6.5 (1.1) | 165.2 ±7.0 (1.0) | 161.8 ±3.9 (1.0) |
| partial update<br>Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows. | 73.5 ±4.9 (1.0) | 93.6 ±5.6 (1.3) | 168.1 ±7.4 (2.3) |
| select row<br>Duration to highlight a row in response to a click on the row. (with 5 warmup iterations). | 7.6 ±4.0 (1.0) | 12.4 ±4.1 (1.0) | 9.8 ±2.5 (1.0) |
| swap rows<br>Time to swap 2 rows on a 1K table. (with 5 warmup iterations). | 20.1 ±4.2 (1.0) | 19.6 ±4.7 (1.0) | 21.8 ±4.5 (1.1) |
| remove row<br>Duration to remove a row. (with 5 warmup iterations). | 46.1 ±2.6 (1.0) | 51.5 ±2.0 (1.1) | 52.5 ±1.8 (1.1) |
| create many rows<br>Duration to create 10,000 rows | 1682.0 ± 53.1 (1.1) | 2033.7 ± 32.0 (1.3) | 1521.4 ± 55.7 (1.0) |
| append rows to large table<br>Duration for adding 1000 rows on a table of 10,000 rows. | 257.6 ±11.1 (1.0) | 271.8 ±9.9 (1.1) | 338.4 ±10.3 (1.3) |
| clear rows<br>Duration to clear the table filled with 10.000 rows. | 360.3 ±16.4 (1.6) | 224.4 ±6.0 (1.0) | 240.9 ±11.4 (1.1) |
| slowdown geometric mean | 1.09 | 1.10 | 1.17 |

*Table 4: Runtime performance benchmarks* (Krause, 2018).

As each of the frameworks applies advanced techniques to make sure runtime performance is excellent which is why performance only differs in orders of milliseconds.

***State Management :***

Each framework offers application state on a component level. In react the ‘useState’ hook is available for functional components and ‘this.state’ is available for class-based components. Furthermore, however React made two recent additions to the library, the useContext and useReducer api which provide more high-level state management options. In Angular data can be attached to the properties of components or state can be maintained within a service and then injected in components. Lastly in Vue state can be stored within the data property of a Vue instance.

You can therefore make any application without an extra state management library. In practice however it is often preferred to manage state outside of components in a separate container because it is easier to scale and maintainable (Banks & Porcello, 2017). To implement the idea of a higher-level state management different state management libraries, exist for each framework. Commonly used once right now are Redux for React, Vuex for Vue and NGRX for Angular. Each of them has in common that they are inspired by the Flux pattern.

Flux is a pattern to manage dataflow throughout an application and its focus is providing explicit and understandable update paths for your application's data. The flux pattern has four nodes: a view, dispatcher, action, and store.

- The view represents the components of an application, it has two main tasks, starting of actions and listening to stores. This means that they just are responsible for triggering data changes and changing UI if this trigger ends up changing the data model.
- Actions can either be triggered from a view (typically a user interaction) or from a different source such as server for example. An action will provide the dispatcher instructions and data required to make a change.
- An action will be dispatched via a central controller called the dispatcher. The dispatcher is designed to queue actions and dispatch them to the right store.

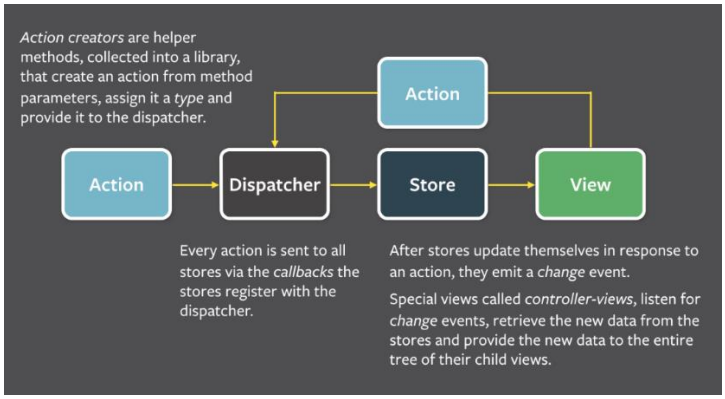



*Figure 21 : Flux structure* (flux, 2019)

The idea is that implementing a one-way data flow ensures that no side effects occur. Data fetching, setting up a subscription, and manually changing the DOM are all examples of side effects (ReactJS, 2020). Instead a store will update data and views will render those updates in the UI. Actions on the other hand describe why changes occur.

***Rendering :***

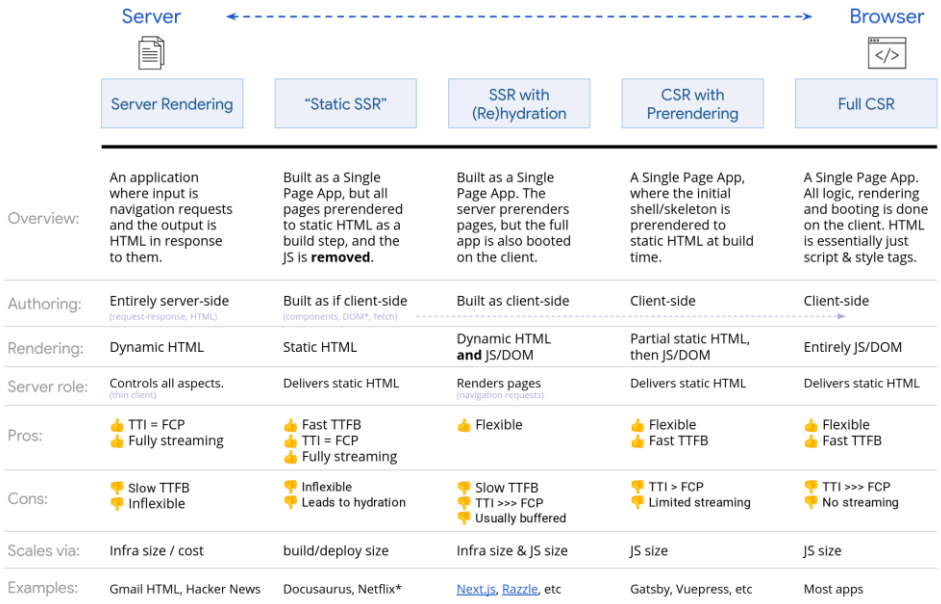



*Figure 22: Possible rendering solutions for web applications* (Miller & Osmani, 2019)

An important decision when developing a web application is determining where it should be rendered (Miller & Adi, Rendering on the Web, 2019) (figure 22). The most common solution implemented by the three JSF is to develop a client-side rendered single page application. A client side rendered SPA will load all application resources into a client's browser upon the first request . Consequently, the entire application will be rendered and executed on the client's device, which enables the UI to be updated on a component basis upon server or user events instead of refreshing the entire page. The main advantage of this approach is that it usually delivers fast user experiences (Mraz, 2019). Additionally, Client-side rendering in web applications can also reduce server loads (Goltzshe at al. 2017).

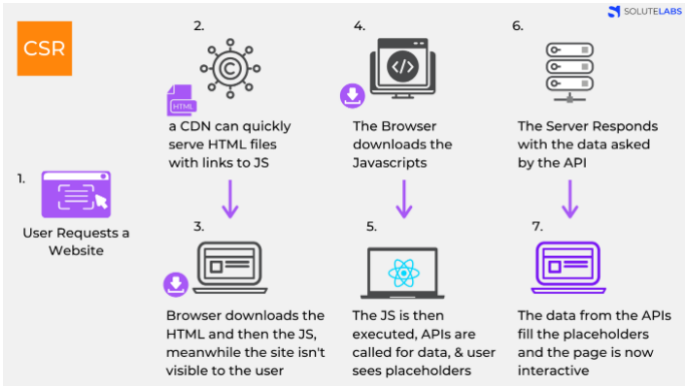



*Figure 23 client-side rendering* (Hanchett & Listwon, 2018).

A common concern with client side rendering however is its suitability for SEO given that HTML content is rendered on the client, web crawlers have trouble scanning the content of a page and thus their SEO rating will be lower. (Madhuri , Balkrishna, & Anushree, 2015). Another concern is that because of a larger bundle size initial load times could increase. This issue is particularly a concern on slower device with a less reliable internet connection.

A way to mitigate some of the downsides of the standard a client-side rendering approach is to use server-side rendering, which is the process of prerendering parts of HTML on the server at runtime. Pre-rendering will ensure that non interactive content is displayed to the user while in the meantime the JS bundle is loaded in the background. All the JSF considered offer a way to achieve this. Vue has a popular higher-level framework Nuxt.js for SSR, Angular offers Angular universal and React SSR is often used with the higher-level framework Next.js.

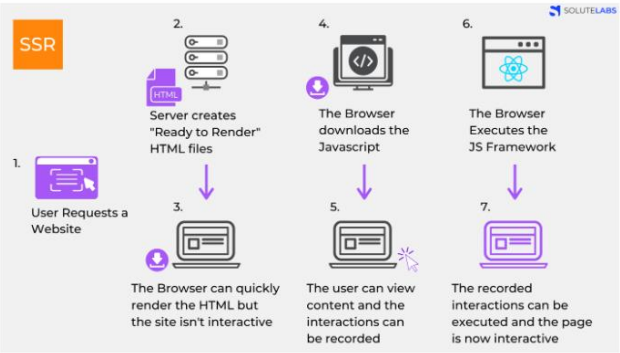



*Figure 24: server-side rendering* (Hanchett & Listwon, 2018).

Another option is to choose static a static site generator. With static site rendering pages will be generated at build time instead of runtime. HTML files are generated once and are then served from the server. This will result will give the fastest possible first page load times. A downside however is that it does not work well with dynamic content because pages need to be made for every possible request which can be a problem if a site cannot predict all possible requests. Static site generators exist for React and Vue the most popular once being Gatsby or Next.js for React.js and Nuxt.js for Vue.js, Angular on the other hand there is no popular static site generator (Miller & Osmani, 2019).

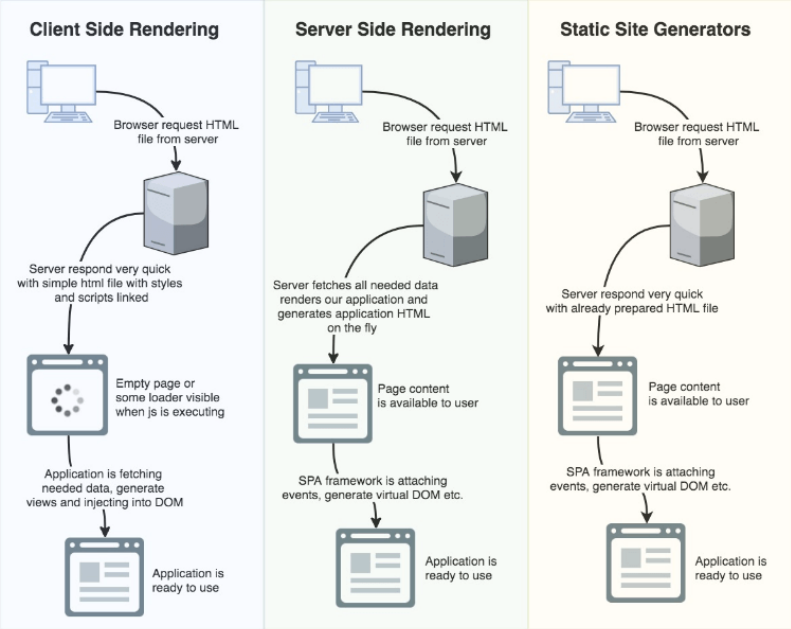



*Figure 25: Comparison of rendering options* (Błaszyński, 2019)

**Non-view related functionalities :**

Angular by default supports development by directly integrating additional modules within the framework, most notably are the inclusion of an HTTP Client, a routing module and a Form validation and handling module. This means that Angular functionalities go beyond support for DOM manipulation Vue on the other hand does not by default include non-view related functionalities but two additional official Vue libraries are available for state-management and routing. React finally does only focus on view-related functionalities and some additional state management support any other functionalities are usually added by importing community supported libraries.

## 6. Methodology

In this paper qualitative research was conducted to develop a quality evaluation framework for front-end JavaScript frameworks. The type of research can be classified as a design science research. Weiringa (2016) offers a template offers the following template for design science within software engineering (figure 26). Within it four question are addressed three questions should be answered.

- What is the problem context? Improve the selection process of a JavaScript framework.
- How is the problem addressed? By implementing the designed artefact, in this case a quality evaluation framework.

- What are the requirements for the artefact? That it can be used within different application and development context.
- To do what? So that practitioners can more easily determine which solution to use within a certain context of use.

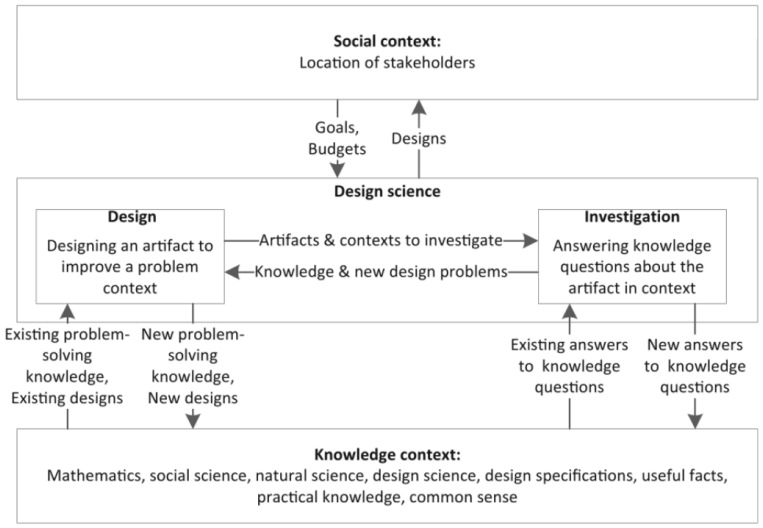



*Figure 26: A framework for design science* (Weiringa, 2016)

In the following a description is given on how this research was conducted to answer our two main research questions. To answer the first research question, a quality characteristics framework was made which applies to the three front-end frameworks. This first version was made by performing exploratory research within the literature study, this included identifying the technical components of front-end JavaScript frameworks by analyzing their documentation as well scientific literature surrounding the frameworks. But also analyzing the non-technical factors which can impact the quality of the frameworks, for this second part most information was drawn from the JavaScript adoption factors proposed by Abrahamson et al. (2018) and the quality evaluation framework proposed by the ISO25000 standards, using an ISO standards ensures that the framework is tested and implemented in a professional environment (Franch & Carvallo, 2003). Next to the benefit of standardization the model remains flexible in its implementation as it can be adapted to different use cases. This is grounded theory research in which a conceptual framework is drawn from the interpretation of data by the researcher.

To ensure the validity of this framework two methods where used. First, triangulation which means that several independent data sources by combining multiple sources ensures a more complete understanding of the research subject as resources can counter or validate each other (Marcos & Lázaro, 2006). Quality requirements are generally inferred from multiple sources which is also the case in this study. Input for the

stakeholder's requirements definition process originates from three sources, technical documentation of the frameworks, scientific literature surrounding the frameworks and scientific literature surrounding quality evaluation frameworks. In the second stage semi-structured interviews with a panel of developers working with the frameworks were conducted, in this case. Conducting these interview falls within the ISO standards in the requirements analysis process (figure 5). To extract insights from the member checking interviews content analysis is performed on each one, this includes breaking down the interviews in smaller units to classify each of them within their relevant category to then derive insights (Joanna & Laplante, 2017).

The interviewees were found by sending out on online survey to developers to probe them for their background, opinion on front-end frameworks and weight that they assign to certain technical and non-technical aspects when choosing a framework, the respondents that were open to an interview where afterwards contacted and an semi structured interview took place in which they were asked to clarify their survey responses and respond to a set of predetermined questions. The survey itself was made up of four main sections which were based upon the initial conceptual framework.

- Section one, questioned the respondents on their background and opinion on frameworks.
- Section two was aimed at clarifying and grading the importance of several non-technical factors for frameworks
- Section three was aimed at clarifying and grading the core functionalities of frameworks
- In the final section participants where free to give their viewpoint on the future of front-end development.

Based upon the responses and upon the initial conceptual framework interview questions were prepared for a group of developers (table 5). Afterwards content analysis of the interview reveals relevancy of each of the categories of the original conceptual framework which was then adapted and modified were necessary.

| Code | Profession | Experience |
|---|---|---|
| P1 | Lead front-end developer at ING France. | 4 years of experience with Angular, 1 year of Experience with React.js. Worked as developer for 8 years |
| P2 | Fullstack developer at a company creating advertising widgets for websites. | Previously used jQuery for several years, for last 5 years focus on React.js |
| P3 | Front-end developer at e-commerce website. | Has 1.5 years of experience with Vue.js |
| P4 | Team leader at e-commerce website. | 3 years of experience with Angular, 1 year of experience with Vue.js and React.js |

| | | |
|---|---|---|
| P5 | Fullstack Developer at Capgemini Paris. | 1 year of experience with Angular. |
| P6 | Front-end developer at Hubx. | 1 year of experience with React |
| P7 | Team Leader at Capgemini Paris. | 5 years of experience with React and respectively Angular |

*Table 5: interviewees (own creation)*

In conclusion, in accordance with ISO 25040, the constructed software quality model and the measures defined are tested by conducting a small case study. This illustrates the measurement process, the process of establishing, planning, performing, and evaluating software measurement within a project. It provides an example of how to adjust the target values to a specific context and how to measure the characteristics defined.

# 7. Study

## *7.1 Requirements definition process*

This process aims to answer research question one. The first conceptual framework is the results of exploratory research conducted within the literature study. Next, the interviews will give an adjusted framework to work with.

Starting with the identification of parts of the ISO model that are relevant in our context of quality evaluation. A custom quality model requires a baseline of properties which are expected and defined (Losavio, Chirinos, Matteo, Lévy, & Ramdane-Cherif, 2004). Our baseline model is defined by mapping the most relevant characteristics found by Abrahamson et al. (2018) to the ISO 25010 defined quality (sub-)characteristics for software quality. As well as by amending the technical characteristics of this framework with the characteristics identified within the literature study.

As mentioned in the literature study the ISO standard has two categories for quality evaluation, software product quality (table 7) and quality in use (table 6). Next to this we have a UTAUT model proposed by Abrahamson et al (table 8) which identifies adoption factors for JavaScript frameworks. Table 9 was constructed considering this as an initial quality evaluation framework. The Functionality category described within table 9 must be viewed next to the functionalities identified for each of the frameworks during the literature study (table 3). The table identifies the functionalities provided by each of the framework and compares them against each other.

| *Characteristic* | *Sub Charachteristic* | *Definition* |
|---|---|---|
| **Context coverage** Degree to which product or system can be used with effectiveness, efficiency and freedom of risk and satisfaction in both specified contexts of use and in contexts beyond. | **Context completeness** | Degree to which product or system can be used with effectiveness, efficiency and freedom of risk and satisfaction in all specified contexts of use. |
| | **Flexibility** | Degree to which a product or system can be used in both specified contexts of use and in contexts beyond those initially explicitly identified. |
| **Effectiveness** | Accuracy and completeness with which accuracy and completeness with which users achieve specified goalsch users achieve specified goals. | |
| **Efficiency** | Resources expended in relation to the accuracy and completeness with which users achieve goals. | |
| **Freedom from risk** | Degree to which the quality of a product or system mitigates or avoids potential risks to economic status, human life, health, or the environment. | |
| **Satisfaction** Degree to which the quality of a product or system mitigates or avoids potential risks to economic status, human life, health, or the environment. | **Comfort** | Degree to which a user obtains pleasure from fulfilling their personal needs |
| | **Pleasure** | Degree to which user or stakeholder has confidence that product or system will behave as intended |
| | **Trust** | Degree to which a user is satisfied with their perceived achievement of pragmatic goals, including the results of use and the consequence of use |
| | **Usefulness** | Degree to which user needs are satisfied when a product or system is used in a specified context of use |

*Table 6: Software product quality in use* (ISO, 2019)

| *characteristic* | *Sub Charachteristic* | *Definition* |
|---|---|---|
| **Functional suitability** Degree to which functions are provided to meet stated and implied user needs under specific conditions. | **Functional completeness** | Degree to which set of functions covers all specified tasks and user objectives. |
| | **Functional correctness** | Degree to which a product or system provides correct results with needed degree of precision. |
| | **Functional appropriatness** | Degree to which the functions facilitate the accomplishment of all specified tasks and objectives. |
| **Performance efficiency** Relative amount of resources used under stated conditions. | **Time behaviour** | Degree to which the response and processing and throughput rates of a product or system, meet requirements when performing it's functions . |
| | **Resource utilization** | Degree to which the amounts and types of resources used by a product or system, when performing it's functions, meet requirements. |
| | **Capacity** | Degree to which maximum limits of a product or system parameter meet requirements . |
| **Compatibility** Degree of software, product, system that can exchange info with other products ,systems and components and still perform it's functions. | **Co-existence** | Product can parform it's functions efficiently while sharing an environment and resources with other products. |
| | **Interoperability** | Degree to which the product can use info provided by others and exchange informtion. |
| **Usability** Degree to which product or system can be used by specified users to achieve specified goals with efficiency, effectiveness and satisfaction in specified context of use. | **Appropriateness recognizability** | Degree to which users can recognize if product is appropriate for their needs. |
| | **Learnability** | Degree on which product or system can be used by specified users to achieve specified goals of learning to use the product or system. |
| | **Operability** | Degree to which product or system has attributes that make it easy to operate and control . |
| | **User protection error** | Degree to which a product or system has attributes that protect user to making errors. |
| | **Ui aesthetics** | Degree to which a a ui enables pleasing and satisying interaction for the user. |
| | **Accessibility** | Degree to which product or system can be used by people with widest range of characteristics and capabilities to achieve a specified goal in a specified context of use . |
| *characteristic* | *Sub Charachteristic* | *Definition* |
| **Reliability** Degree to which a system or product or components performs specified functions under specified conditions for a specified period. | **Maturity** | Degree to which system product or component meets reliability needs under normal operation. |
| | **Availability** | Degree to which a system or product or comonent is operational and accesible when required for use. |
| | **Fault tolerance** | Degree to which a system product or compoentn is operational and accesible despite presence of hardware or software faults. |
| | **Recoverability** | Degree to which a system, product or component is operational and accesible when required for use. |
| **Security** Degree to which a system, product or component is protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authority. | **Confidentiality** | Not taken into consideration |
| | **Integrity** | |
| | **Non-repudiation** | |
| | **Accountability** | |
| | **Authenticity** | |
| **Maintainability** Degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers. | **Modularity** | Degree to which system or program is made of discrete components such that change in one has minimal impact on the other. |
| | **Reusability** | Degree to which an asset can be used in more then one system or in building other assets. |
| | **Analysability** | Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failure, or to identify parts to be modified. |
| | **Modifiability** | Degree too which a product can be effectively and efficiently modified without introducing defects or degrading existing product quality. |
| | **Testability** | Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met. |
| **Portability** Degree of effectiveness and efficiency with which a system product or component can be transferred from one hardware, software or usage environment to another. | **Adaptability** | Degree of effectiveness and efficiency with which a system product or component can adapt to different or evolving hardware software or other operational or usage environments. |
| | **Installability** | Degree of effectiveness and efficiency with which a product or system can be succesfully installed and or uninstalled in a specific environment. |
| | **Replaceability** | Degree to which a product can replace another specified software product for the same purpose in the same environment. |

*Table 7: Software product quality characteristics* (ISO, 2019)

| *Main JSF adoption factor* | *Sub JSF adoption factor* | *Implication* |
|---|---|---|
| **Performance expectancy** Gains in product performance not in developers productivity, this would be categorized under effort expectancy | **Performance** | - A framework [documentation, author, website] should state clearly if it is intended for developing applications that will run relying mostly on the resources of the client or on those of the server. |
| | **Size** | - The number of lines of code of the applications created by means of the framework should be as low as possible. |
| **Effort expectancy** | **Learnability** | - The requirements for starting to use a framework fast can be high or low but this also depends on the technical skills of the developer and the time to submit a project. |
| | **Understandability** | - Documentation should be precise and include several examples for implementing common tasks. Documentation should allow developers to find guidelines on implementing a feature quickly.<br>- The code that implements a framework should be easy to read and understand. At the same time, a framework should allow the creation of cleaner code as well.<br>- A framework should enable the development of clear code. |
| | **Complexity** | - Perceived reduction of complexity while developing is beneficial. |
| ***social influence*** | **Competitor analysis** | - A framework is estimated as reliable if used by similar companies in their production environment. |
| | **Collegial advice** | - The code represents those direct recommendations from peers and self-constructed networks of trust that practitioners employ when considering a framework. |
| | **Community size** | - The community behind a framework is its heart and as such its size and contributors signal the trust of developers in the technology. |
| | **Community responsiveness** | |
| **facilitating conditions** | **Suitability** | - Simple tasks such as event handling, DOM manipulation, and real time component updates should be automatized. |
| | **Updates** | - Frequently updating a framework with new features for matching web design trends is evaluated positively. |
| | **Modularity** | - A framework should be modular; changing a module should not affect others. The modularity of a framework means that it is flexible, too. It should be easy to import libraries and features.<br>- Libraries that come with a framework should enable the achievement of basic and advanced functionalities at any development stage. |
| | **Isolation** | - A framework should encapsulate logical components |
| | **Extensibility** | - A framework should allow external libraries to be imported without having to adapt them |

*Table 8: JSF adoption factors* (Graziotin, Abhramsson, & Pano, 2018)

| *Category* | *Charachteristic* | *Definition* |
|---|---|---|
| **Functionalty** The intrinsic technical aspects of the framework | **Functional completeness** | Degree to which the framework provides required functionalities for a given application |
| | **Functional performance** | Degree to which the functionalities perform efficiently and effective |
| | **Modularity** | Degree to which a framework is open for importing other libraries of features |
| | **Portability** | Degree to which a framework can be adopted to different use cases |
| **Ecosystem** The non technical aspects surrounding the framework | **General Community** | Degree to which a framework is supported by a large and responsive community |
| | **Proffesional Community** | Degree to which similar applications adopt a framework or not, and degree to which colleagues adopt the framework |
| | **Documentation** | Degree to which the framework offers information on it's usage and implementations |
| | **Updates** | Degree to which the framework is maintained and updated |
| **Effort** Charactzeristics of a framework which define ease of adoption for a developer | **Learnablity** | Time it takes to understand and implement a framework. |
| | **Complexity** | Degree to which the developer perceives that the framework reduces application complexity |

*Table 9: Initial framework (own creation)*

### *2 Requirement analysis process*

To adjust this initial model an interview was conducted with seven different developers (table 5). The interview consisted of a set of questions based upon the initial conceptual framework which were the same for every interviewee (table 10) and a set of questions based upon their survey responses.

| *Category* | *Charachteristic* | *Questions* |
|---|---|---|
| **Functionalty** The intrinsic technical aspects of the framework | **Functional completeness** | What technical functionalities do front-end framework provide for your application and when do you consider them essential? |
| | **Functional performance** | How important is the performance of a framework for you and what do you consider a well performing framework? |
| | **Modularity** | How flexible do you expect a framework to be in adopting external libraries when used within an application? |
| | **Portability** | How important is it for you have a framework that can be used in a variety of use cases such as mobile development, server side rendering, static site generation etc. ? |
| **Ecosystem** The non technical aspects surrounding the framework | **General Community** | What do you consider a good community and when is this important to you? |
| | **Proffesional Community** | Is it important to see that a framework is being widely adopted and used by similar applications? |
| | **Documentation** | How important is the support from the framework for you in terms of documentation? |
| | **Updates** | Do you see updates as an important factor when choosing a framework? |
| **Effort** Charactzeristics of a framework which define ease of adoption for a developer | **Learnablity** | Do you expect a framework to be easy to pick up and why? |
| | **Complexity** | In what aspects does a framework reduce application complexity according to you? |

*Table 10: Questions concerning initial framework. (own creation)*

# Interview Content Analysis

## 1) Functionality:

Interviewees saw DOM manipulation as the essential part for each of the frameworks. Next to this they also stated that a component-based approach is the future for web development. For additional functionalities such as rendering solutions, and state management however most agreed that this is highly context dependent. The two main contextual factors are the application type and the respective team in which they are developing. Depending on these two aspects a framework may or may not meet the standards in terms of functional completeness.

Primary influential application factors:

- **Target platform:** If an app is developed for the browser, mobile or desktop.
  - ➔ SSR is an important feature to decrease load time on mobile
- **State Complexity:** The volume and mutations of state saved on front-end.
  - ➔ The more data involved in the application the more valuable state management becomes.

*P1: "Frameworks can be overkill for simple sites in that case you are better of using something like web components because those only lack the state management and reactivity"*

- **Content:** Is the content of the application aimed at interaction or documentation.
  - ➔ Server-side rendering is important if content needs to be SEO friendly.
  - ➔ Document oriented increases attractiveness of static site generation.
  - ➔ DOM manipulation and CSR is important when interaction is high.

*P7: "Actually clients only care about SSR when they are worried about SEO"*

*P3: "The hardest thing to do is creating a reactive app if an app is non-reactive you actually don't need a framework"*

Primary influential team factors:

- **Size:** How many developers are working on the same project
  - ➔ as more developers join a project a clearly defined architecture is seen as better.

  *P1: "Angular can be better as we don't have to deal with configuration, and we are sure that things are stable"*

  *P6: "Opinionated is good in bigger teams because it ensures coherency"*

  *P5: "We are with a small team of two so Vue is perfect, we can move fast."*
- **Experience:** Experience of developers.
  - ➔ More experience will make less opinionated better

  *P5: "The team I work in and the type of app have the most influence. If I have more senior developers React will be more interesting. If I need to move fast within strict guidelines Angular is interesting"*

**Implication**: DOM manipulation and a component-based approach are essential functionalities for frameworks. Additional features become important dependent on application type and development team.

## 2) Performance:

In terms of performance developers did consider this as very important, however they also consider these frameworks to be performant by default given their size and support. Most of the developers believe their performance is so similar that for most applications it will not be a determining factor. In the case however in which you might use one of them on mobile the once with a smaller bundle size might be favorable given a faster load time.

*P1: "When app is reactive even then server response time will be of higher importance then the framework its DOM manipulation capabilities"*
*P2: "A small bundle size can be important if served under lower network speed and slower devices"*
*P3: "Widely used frameworks will have an acceptable performance in most use cases."*
*P4: "Performance is not that important for me; I consider on time delivery more of a concern for me."*
*P5: "Performance is a question of milliseconds and not relevant in most cases"*

**Implication:** A frameworks performance is generally expected. When it comes to load times however a more significant difference is possible given the differences in modularity.

## 3) Modularity:

In terms of modularity developers stated that in some cases this could be a pro and in others a con. This again is a factor that is dependent on the context. More specifically it was stated that a reduction in modularity and an increase in structure within opinionated frameworks can be a pro when working in a team to ensure consistency or when working with younger developers to enable them to be productive more quickly.

Primary influential team factors:

- **Size:** How many developers are working on the same project
  ➔ Negatively correlated with more modularity.
- **Experience:** Experience of developers.
  ➔ More experience will make modularity positive

*P1: "When you have more seniors in your team React becomes more interesting because they can deal with configuration better"*
*P2: "Vue can be better if you want to involve more of a design team because it is more basic and less complex"*

**Implication:** Modularity is considered better under a smaller and more experienced a team.

## 4) Portability:

When it comes to portability most of the developers considered each of the frameworks to be versatile enough as they all offer several options for server-side rendering or mobile development. The ecosystem for these solutions however was considered better for react given it is more widespread use in for example higher level framework like Gatsby or next or even mobile development with React Native. When it comes to client side rendered single page applications however, they were all considered to be sufficiently adaptable to different demands. For SSR and static site generation Angular was a bad option given that there are no higher-level static site generators for it and the low popularity of Angular universal.

Primary influential application factors:

- **Target platform:** If an app is developed for the browser, mobile or desktop.

**Implication:** Portability is only an important factor if the application should run out of the browser.

> *P5: "React native is interesting if you have need for a mobile application and don't have the resources for a native application otherwise It doesn't really matter that much"*

## 5) Community:

A good community is one that is mainly responsive which usually comes with size. For most developers, the main indicator for a responsive community is the response rate and question count on stack overflow. Many questions being an indication of a large community and a high answer rate indicating a responsive community. When it comes to professional communities the interviewees indicated that this does not signify quality for them.

*P7: "Indication of how good a community is, is usually seen on stack overflow which gives a real representation"*
*P2: "questions have to be answered otherwise a large community is useless"*
*P3: "Importance of community increases if you are in a smaller team and must rely extra on online resources"*

**Implication:** Community is an important factor because it will lead to better support and faster development with a certain framework. A good indicator for communities is stack overflow.

## 6) Documentation:

Documentation was important but rather it is something that can fall under community because with a good community good documentation will grow. When it came to official documentation, developers stated that the quality and structure is very similar  thus it will not have a big impact on their choice.

**Implication:** Documentation can be left out as an impacting factor given that developers will rather go forth on community

## 7) Updates:

Fast addition of new features to a framework is a big plus because in the current front-end development framework everything evolves fast frameworks are obligated to follow suit.

*P2: "Some libraries are unreliable because of their big amount of issues I think these frameworks don't have that problem because of their large backing"*
*P7:  "Of course I expect these big players to always have the latest features integrated in their tools"*

**Implication:** The frameworks are expected to be continuously maintained and updated with new functionalities.

## 8) Learnability:

Most developers stated that over time they can learn any technology. The only factor which makes learnability important is time for delivery, if an app needs to be delivered fast it was an advantage to be easy to learn.

*P2: "I do not think that it is that relevant in the beginning because you can grow understanding over time."*
*P6: "The time to get started is important if we can get started fast without a high barrier that is a good KPI."*
*P7: "Learnability would only be important if you have newcomers and you want to get them up and running quickly "*

**Implication:** learnability is seen as an important factor only when there is a time-constraint.

## 9) Complexity:

Frameworks can reduce complexity and increase standardization when implemented correctly.  Developers agreed that when apps become increasingly complex it is almost impossible to keep code manageable without a framework. To have a simple framework structure is thus considered essential

*P6: "if you start with vanilla JS and your app gets complexity added to it things can get messy quickly."*

**Implication**: Frameworks should aim at  abstracting away application complexity and should deliver readable and understandable code

Considering the insights made in the interviews the framework was adapted to be more context dependent. Requiring specification of both team and application type as well as leaving out less relevant characteristics. Figure 27 includes the adjusted context dependent quality evaluation frameworks.

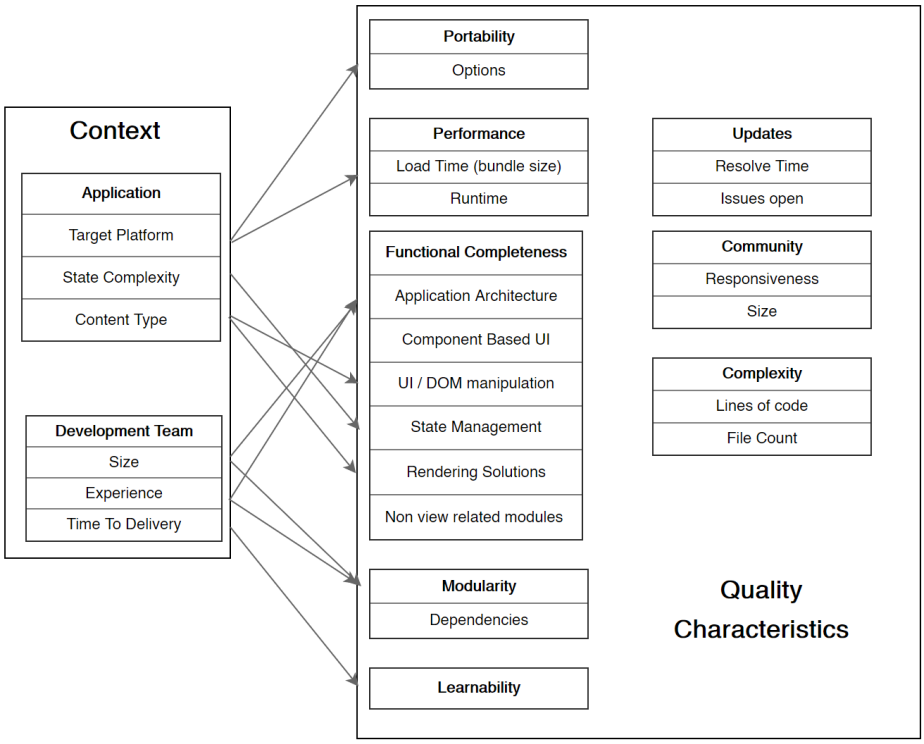



*Figure 27: Context dependent quality evaluation framework*

### *7.3 Operationalization*

For the framework to be applicable on a project the characteristics identified must  be operationalized into indicators that reflect to what degree a product satisfies a characteristic. In the table Ut infra, indicators are proposed for a subset of the characteristics. The once for which no measurement will take place however are learnability, state complexity and run-time performance. Run-time performance is not measured because in most cases it is expected to be only a small difference between the frameworks. State complexity and learnability gets no indicator because of their subjective nature but this does not mean that indicators could not be proposed in the future.

| | | |
|---|---|---|
| **Application** | **Target Platform** | If the application is intended run in the browser, native on mobile or on desktop. |
| | **Content type** | Is application content documentation – or interaction oriented. |
| **Development Team** | **Size** | Number of developers working on the project. |
| | **Experience** | Measured as number of years of professional experience with front-end development.<br>- Negatively correlated with a more opinionated application architecture,<br>Positively correlated with more modularity. |
| | **Time to deliver** | Days to until application must be delivered.<br>Positively correlated with Learnability as a quality characteristic. |
| **Portability** | **Options** | Number of optional platforms on which you can develop with the frameworks |
| **Performance** | **Load time** | Time it takes to have the application ready for interaction on a client's device, determined by bundle size. |
| **Functional completeness:** | **Items covered** | Functional areas of table 3 that are necessary and covered. Depending on the context presence and quality of one functionality can be more important than that of another. |
| **Modularity** | **NPM packages dependencies** | NPM packages depended on and thus made for the specific frameworks. |
| **Updates *** | **Resolution time** | The resolution time is the median time an issue or pull request on GitHub Stays open (Appendix 1). |
| | **Open percentage** | The percentage of open issues and pull requests on GitHub (Appendix 2). |
| **Complexity *** | **Total Lines of Code** | Total lines of code within per JavaScript file source directory. |
| | **Average lines of Code per file** | Total lines of code per JavaScript file within the source directory. |
| | **Maintainability index** | Index which signifies maintainability of a repository. |
| **Community*** | **Stack overflow tag statistics** | How responsive and big is a community. Stack overflow serves as indicator for both size and responsiveness. (Appendix 3 and 4). |

*Table 11: proposed indicators (own creation)*

** Complexity: Will be measured using the tool Plato* (Overson, 2015). *Plato is a NPM package which is able to parse source-code and output the metrics defined above and output a visual report on the statistics for each repository* (Mariano, 2017).

** Community: Statistics about community were retrieved using queries defined in appendix 3 and 4*

** Updates: Average resolution time and open percentage were retrieved using functions defined in appendix 1 and 2*

### *7.4 Implementation*

**Setup:**

The setup for the app is quite simple, it is a to-do application using a node.js backend and on it you can login in or signup and consequently create, read update, and delete personal to-dos with the provided UI either written in React, Angular or Vue. To create the frontend architecture the CLI's of each of the frameworks were used respectively to create the frontend setup the command used are ´vue create <project-name>´, ´ng new <project-name>´, ´create-react-app <project-name>´. Each provides all necessary configuration and setup to start development immediately. The application's frontend is visually identical the only difference being that they each run on a different framework. For the backend, a node API build with express is used together with a mongo DB database in where all users and to-dos can be stored at. In the end all the applications where also deployed to Heroku on the following links (source code : https://github.com/henridev/thesis-frontend-frameworks). The sites are deployed on following URLs: https://angular-thesis.herokuapp.com/, https://react-thesis.herokuapp.com/, https://vue-thesis.herokuapp.com/.

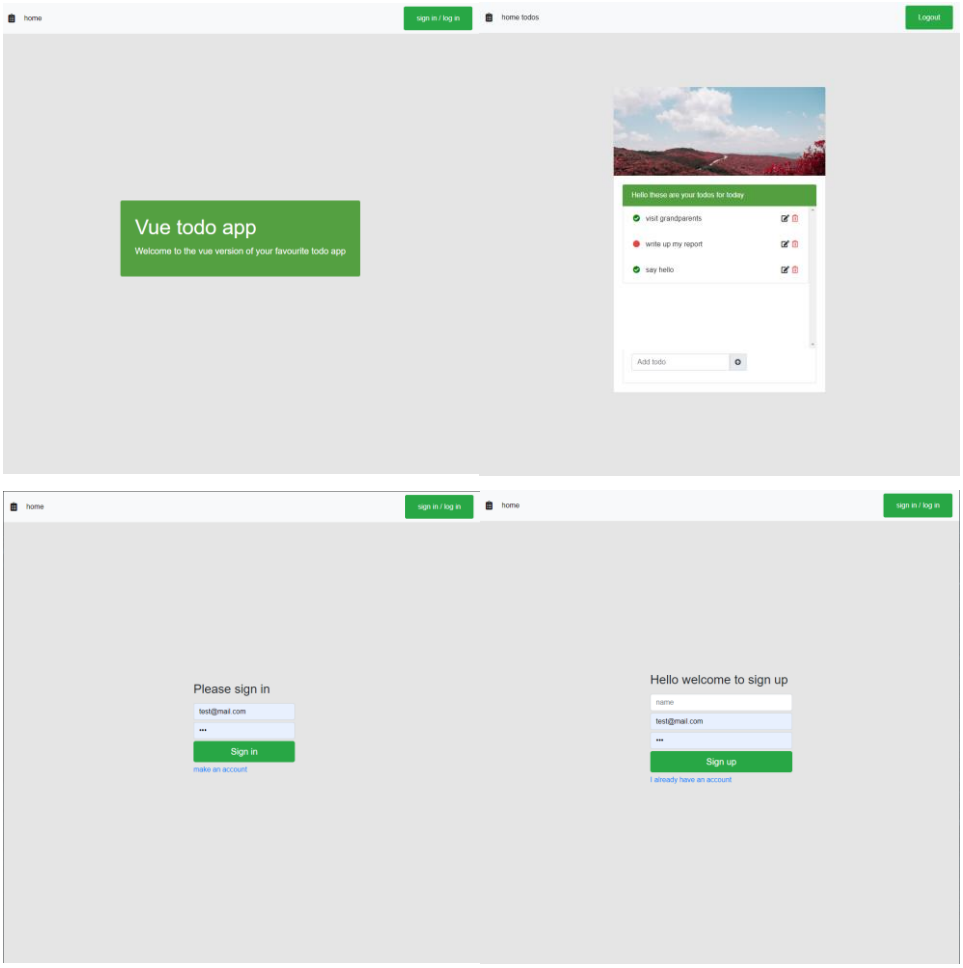



*Figure 28: Application screens (own creation)*

**Context:**

In accordance with the adapted quality evaluation framework (figure 30) both Team and application factors are considered. Our application is meant to be developed for a web browser and has a low level of Data Complexity, only to-dos and users need to be saved which does not involve much state-management. Next to this the app will be application instead of documentation oriented.

In terms of team size only on person will be working on the app with around 1 year of experience with frontend Development. The time to deliver is not relevant as it is not in a professional context. As a result of this context portability and learnability will not be taken into consideration as indicators. In terms of performance runtime are expected to be similar so only bundle size is compared.

**Indicators:**

1. **Functional completeness**:

| Functionalities | React todo app | Angular todo app | Vue todo app |
|---|---|---|---|
| ***UI architecture*** | Component based | Component based | Component based |
| ***UI / DOM manipulation*** | Virtual DOM | real DOM | Virtual DOM |
| ***State Management*** <br> **** each is extensible with other state management libraries*** | Only use of component level state | Only use of service level state | Only use of component level state |
| ***Rendering Solutions*** | Standard CSR SPA | Standard CSR SPA | Standard CSR SPA |
| ***Non-view related modules*** | - import Axios as extra HTTP client <br> - import community library React router for routing | Includes HTTP Client, routing module and form handling and validation module | - installed Axios as extra HTTP client <br> - import core library Vue router for routing |

*Table 12: Functional aspects of application (own creation)*

Angular required no installation of external libraries as all needed functionalities where already included for Vue and React however additional libraries where added both for routing and HTTP requests. For react this meant importing a community library for vue it meant importing a core library.

2. **Performance**: Bundle size after performing compilation with webpack to convert source-code to vanilla JS code was analyzed by using webpack-bundle-analyzer which returns total gzipped bundle size for each front-end implementation.

After compilation via webpack Angular has a gzipped bundle size of 132.51 kb in comparison to 56.33kb for React and 52.9 kb for Vue.js. Vue and React will thus be faster to load onto a client device in this case. This is as expected given that Angular is seen as the more inclusive framework while React and Vue aim to be lightweight. For this use case React and Vue are thus a better choice performance wise.

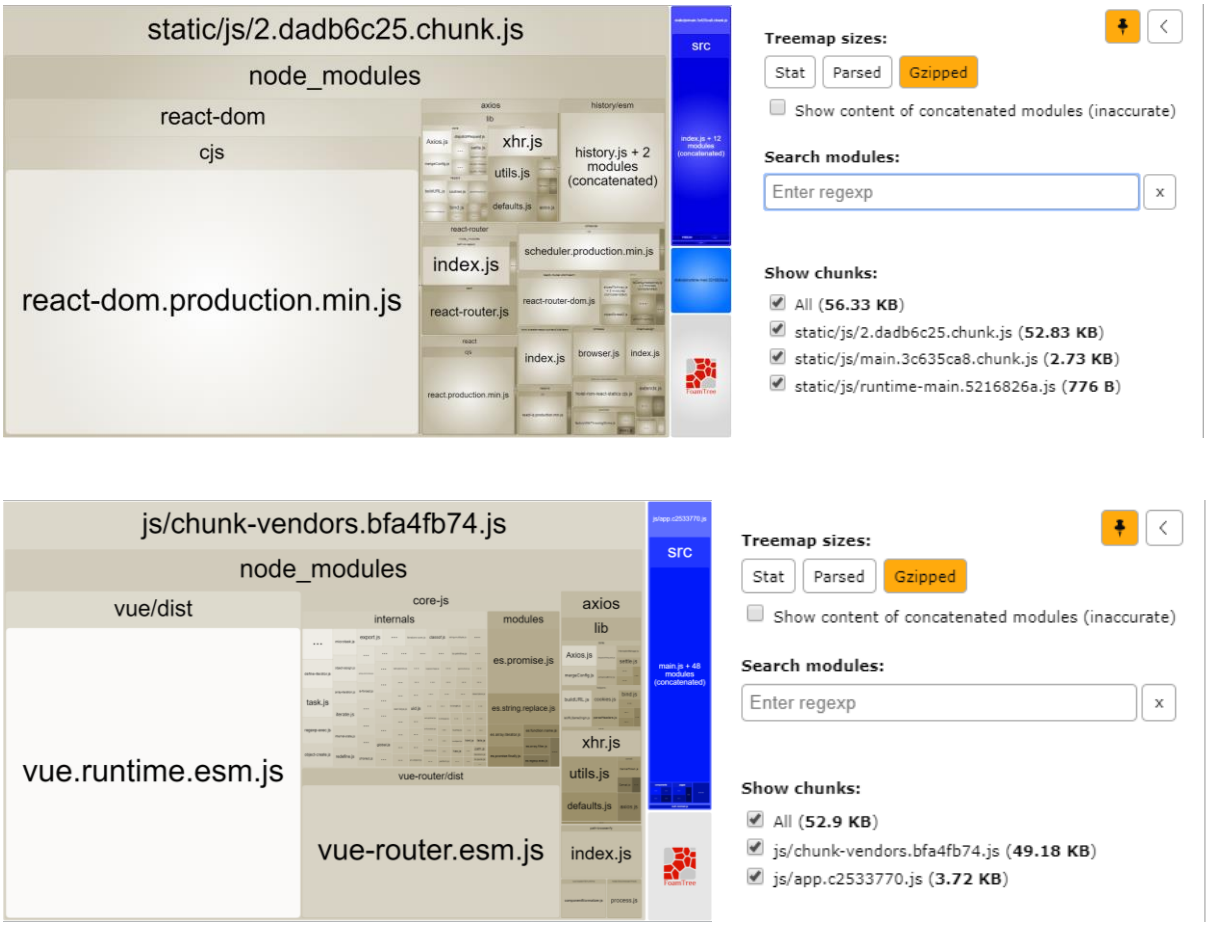



*Figure 29: Bundle size for each front-end framework (own creation)*

3. **Modularity:** NPM packages depending on and thus made for the specific frameworks.

Angular is the least modular of the three as it offers the lowest amount of NPM modules to integrate within a project . React on the other has 56661 modules depending on it meaning it offers the largest module ecosystem of the three. Vue falls in the middle with 27544 dependents. In general, Angular offers inclusiveness while React and Vue offer more options for modularity.

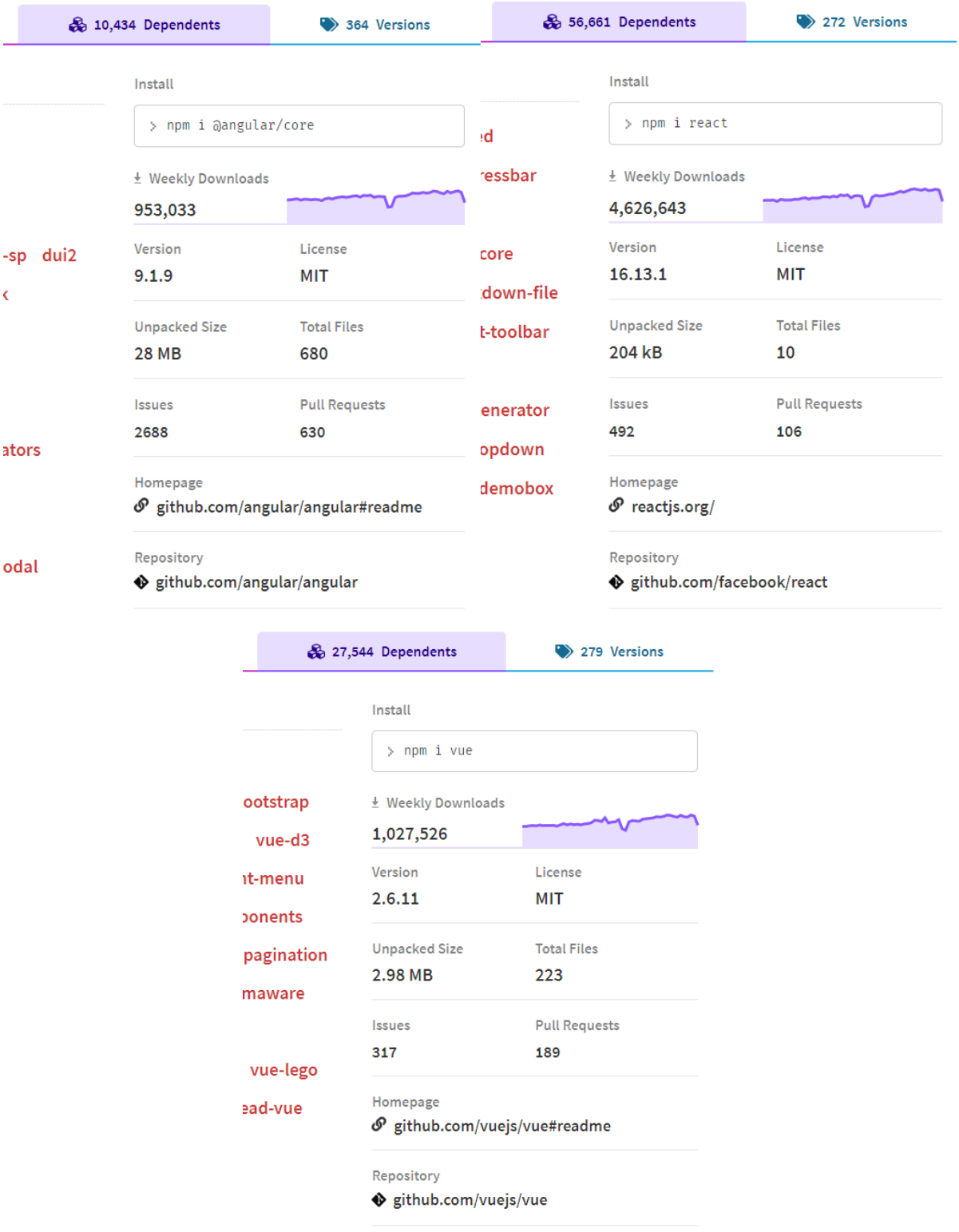



*Figure 30: NPM packages available for each front-end framework (own creation)*

4. **Updates:** Indicated by the average resolution time of GitHub issues and rate of open issues.

| | Vue | React | Angular |
|---|---|---|---|
| Median resolution time | 45 minutes | 21 hours | 11 days |
| Open issue percentage | 1% | 15% | 11 % |

*Table 13: Indicators for updates (own creation)*

The Vue core team provides the fastest resolution speed and lowest percentage of open issues. Meaning that in comparison to React and Angular developers can be more certain that any problems encountered in the framework will be resolved quickly.

## 5. Community

Size is indicated by total count of stack overflow posts indicated with the frameworks tag. Responsiveness is indicated statistics surrounding the posts. Angular and React have an advantage over Vue here as they have greater size which is almost equal. In terms of responsiveness React has an advantage as it has the lowest total time to answer. Vue on the other hand performs the worst of the three as at it has the lowest responsiveness and size

| TagName | N | median_views | avg_score | closed_rate | avg_answers | accepted_rate | answer_rate | median_tta | Reputation |
|---|---|---|---|---|---|---|---|---|---|
| reactjs | 211918 | 138 | 1,60 | 2,2 | 1,2 | 44,5 | 79,6 | 25 | 2016,19 |
| angular | 215861 | 260 | 1,70 | 2,3 | 1,3 | 43,8 | 79,8 | 30 | 2039,07 |
| vue.js | 56490 | 167 | 1,30 | 1,6 | 1,1 | 45,7 | 78,5 | 45 | 1804,78 |

*Table 14: statistics on stack overflow questions per framework after 2018 (Appendix 3 and 4)*

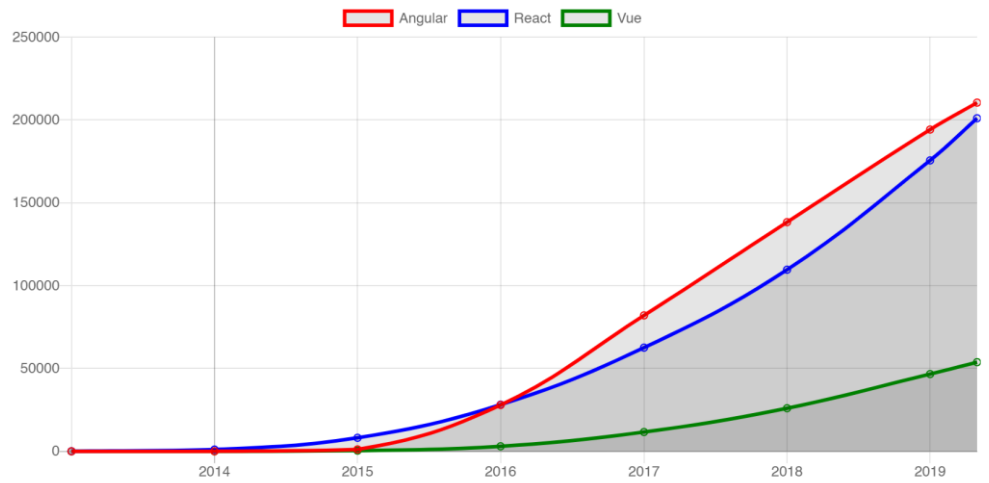


*Figure 31: Total stack overflow posts tagged with framework by year (own creation)*

## 6. Complexity:

Complexity was measured by using the Plato complexity measurement tool (Overson) on each of the source directories of the applications the tool performed the analysis for which the resulting reports are included in appendix 5. The resulting maintainability index is based on the following formula:

$$Maintainability\ Index = 171 - 5.2 * ln(Halstead\ Volume) - 0.23 * (Cyclomatic\ Complexity) - 16.2 * ln(Lines\ of\ Code)$$

The Angular application received the highest maintainability score while maintaining the lowest lines of code per file. React on the other hand performed the worst by having the highest total and average lines of code while also having the lowest maintainability score. Vue on the other hand performed very well on total lines of code because of its close resemblance to vanilla JS the compiling to JS files had a positive impact on the total lines of count.

| | **Vue** | **React** | **Angular** |
|---|---|---|---|
| **Total lines of code** | 127 | 1092 | 777 |
| **Average lines of code** | 31 | 64 | 25 |
| **Maintainability index** | 81.13 | 71.36 | 78.75 |

*Table 15: Indicators for complexity (own creation)*

**Conclusion:**

| | Vue | React | Angular |
|---|---|---|---|
| **Functional completeness** | Core library vue-router and community developed library | Community developed libraries react router and Axios imported. | No aditional libraries required |
| **Performance** | 52.9kb bundle size | 56.33kb bundle size | 131.51kb bundle size |
| **Modularity** | 27544 modules | 56661 modules | 10434 modules |
| **Updates** | Median resolution time 45 minutes | Median resolution time 21 hours | Median resolution time 11 days |
| | Open issue percentage 1% | Open issue percentage 15% | Open issue percentage 11% |
| **Community responsiveness** | median TTA 45 | median TTA 25 | median TTA 30 |
| **Community size** | 56490 stack overflow questions | 211918 stack overflow questions | 215861 stack overflow questions |
| **Total lines of code** | 127 | 1092 | 777 |
| **Average lines of code** | 31 | 64 | 25 |
| **Maintainability index** | 81.13 | 71.36 | 78.75 |

*Table 16: Comparison table of results for quality characteristic indicators (own creation)*

Each framework comes with its own set of quality characteristics. While this table gives an indication of which frameworks performs best on each of the characteristics within this use case it is still up to the developer to determine how much they value each of the characteristics and to what extend they apply to their particular use case. For this example, Vue was the best solution as it provided a non-complex solution for a simple project which is what was valued more in this case in comparison to the other characteristics.

# 8. Results

## *8.1 limitations and scope*

Due do the big variety of client-side JS Frameworks that are available, it was impossible to include all of them within this research, which is why a popular subset was chosen. Although it is important to recognize that there is a plethora of other options out there. Furthermore, when it comes to measuring the characteristics identified it is hard to find indicators with high validity for a given characteristics. Therefore, this was rather an attempt to identify the most important and correct ones. Also, for measurement it must be acknowledged that the sample application is by no means a correct generalization on frameworks quality in general but rather an estimation of how they compare within a given context. For these two reasons extra research is necessary to verify validity and reliability of indicators.

Also, to maintain a manageable scope for this paper not all the targets and stakeholders were considered. It is however important to recognize it is that the quality evaluation of frontend frameworks is dependent on a larger system. This could be the application it is used in, the computer system it is used in, the information system of the business developing the application or even the business system (figure 32) (ISO, 2019). In this specific context however, the primary focus is on the target software itself which is the frontend framework and the application in which it is used as well as the usage context outside the application. In the study one stakeholder is taken into consideration, the developer's perspectives. Given that they are the primary users of the product who are in contact with it on a regular basis.

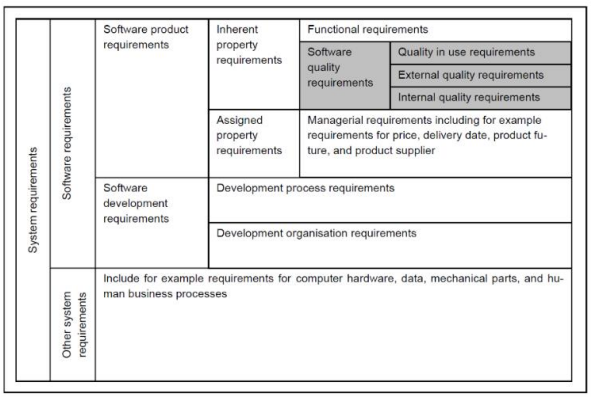



*Figure 32: Targets of quality models* (ISO, 2013)

In the future research could be expanded to target different frameworks. Next to this the model's validity could be verified on a larger scale by conducting a statistical analysis via surveys to determine correlation between elements.

### *8.2 Discussion*

Research question one uncovered that the front-end frameworks Vue.js, Angular and React.js can be characterized on, Syntax, Application architecture, UI architecture, UI/DOM manipulation, state management, non-view related functionalities and rendering solutions. Table 3 provides an overview of how each of the frameworks compare to one another on the most relevant characteristics.

For the second question a quality evaluation framework was developed by including the most relevant characteristics proposed by developers and derived from a literature study. It shows that two major contextual factors to consider when adopting a front-end JSF are Team and Application, as they will influence if certain

frameworks characteristics are of a positive or negative influence.  Next to this the most relevant characteristics to consider where identified, portability, performance, Functional Completeness, Modularity, Learnability, Updates, Community, and Complexity.

Finally, indicators for the characteristics are proposed and applied to each of the frameworks for a small sample application. It shows that in practice this framework can help practitioners identify and characteristics that are of importance to them when choosing a front-end JSF.

## 9. Conclusion

Right now, the use of frameworks is very prevalent within front-end web development nonetheless most developers follow intuition and general knowledge when choosing one. As was previously identified however frameworks possess certain characteristics which will increase their adoption rate (Graziotin & Abhramsson, 2018). This research expands on that in that it validates if these adoption factors apply for current most popular frameworks. On top of it, the ISO SQUARE standards where used to identify, operationalize, and evaluate the characteristics. Offering a tool for practitioners on which they can, rely to find the most suitable solution for their problem.

This paper offers several contributions. First, it helps practitioners choose a framework best suited to their needs and stated requirements. Second the findings in the paper can be used for future research. Third findings of these paper can be serving as input for the developers of frontend frameworks so that new frameworks or evolutions of current frameworks align better with their user needs and requirements.

# Bibliography


Koretskyi, M. (2018, 10 08). What every front-end developer should know about change detection in Angular and React. Retrieved from indepthdev: https://indepth.dev/what-every-front-end-developer-should-know-about-change-detection-in-angular-and-react/

Abran, A., Qutaish, R. A., Habra, N., & Desharnais, J.-M. (2005). An information model for software quality measurement with ISO standards. Metrics News Journal , 35-44.

Amjad, M., Hanaan, A. S., & Moiz, A. U. (2017). Introduction to Web Development. Islamabad: Commonwealth of Learning.

Andreas B. Gizas, S. P. (2013). Comparative Evaluation of JavaScript Frameworks. 21st international conference companion on World Wide Web, (pp. 513-514).

Angular. (2020). Retrieved from https://angular.io: https://angular.io

Aquinno, C., & Gandee, T. (2016). Front-End Web Development: Guide. Atlanta: The Big Nerd Ranch.

Banks, A., & Porcello, E. (2017). Learning React Functional Web Development with react and redux. Sebastopol: O'Reilly.

Bibeault, B., & Kat, Y. (2008). jQuery in Action. Greenwich: Manning Publications Co.

Błaszyński, Ł. (2019). Client and Server Side Rendering Static Site Generators. Retrieved from espeo software: https://espeo.eu/

Boduch , A., & Fedosejev, A. (2017). React 16 Essentials: A fast-paced, hands-on guide to designing and building scalable and mainainable webapps with react 16. Birmingham: Packt Publishing.

Brown, E. (2019). Web Development with node and express, Leveraging the JavaScript Stack. Sebastopol: O'Reilly.

Burbeck, S. (1992). Applications programming in smalltalk-80: how to use model-view-controller (mvc).

Christoph, R. A., & Majchrzak, T. A. (2019). Towards the definitive evaluation framework for cross-platform app development approaches. The Journal of Systems and Software 153, 175–199.

Curcio, K., Malucelli, M., Reinehr, R., & Antônio, M. P. (2016). An analysis of the factors determining software product quality: A comparative study. Computer Standards & Interfaces, 10–18.

David Goltzsche, C. W. (2017). TrustJS: Trusted Client-side Execution of JavaScript. the 10th European Workshop on Systems Security (EuroSec'17).

Dayley, B. (2014). Node.js, MongoDB, and AngularJS Web Development. Pearson Education.

Dayley, B. (2017). Node.js, MongoDB and Angular Web Development: The definitive guide to using.

DeBill, E. (2020, february 28). module counts. Retrieved from modulecounts: http://www.modulecounts.com/

del Pilar Salas-Zárate, M., Alor-Hernández, G., Valencia-Garcia , R., Rodriguez-Mazahua, L., Rodriguez-Gonzalez, A., & Cuadrado, J. L. (2015). Analyzing best practices on Web development frameworks:. Science Of Computer Programming, 1-19.

Dominguez-Mayo, Escalona, Meijas, & Staples. (2012). Quality Evaluation for MOdel-Driven Web Engineering methodologies. Information and software technology.

Ericson, J. (2017). Retrieved from stackExchange: https://data.stackexchange.com/stackoverflow/query/1225071/question-stats-by-tag

Fink, G., & Flatow, I. (2014). Pro Single Page Application Development: Using Backbone.js and ASP.NET. Apress.

flux. (2019). Retrieved from facebook.github.io: https://facebook.github.io/flux/docs/in-depth-overview/

flux. (2019). Retrieved from github: https://github.com/facebook/flux

Franch, X., & Carvallo, J. P. (2003). Using Quality Models in Software Package Selection. IEEE Computer Society, 34-40.

Galli, M. (2019, 08 29). mdn docs. Retrieved from MDN web docs: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Traversing_an_HTML_table_with_JavaScript_and_DOM_Interfaces

Graziotin, D., & Abhramsson, P. (2018). Factors and actors leading to the adoption of a JavaScript Framework. Empirical Software Engineering vol 23, 3503-3534.

Graziotin, D., & Abrahamson, P. (2013). jungle of JavaScript frameworks. Product Focused Software Process Improvement (pp. 334-337). Paphos: 14th internationale conference.
Graziotin, D., Abhramsson, P., & Pano, A. (2018). Factors and actors leading to the adoption of a JavaScript Framework. Empirical Software Engineering vol 23, 3503-3534.
Hanchett, E., & Listwon, B. (2018). Vue.js in action. New York: Manning Publications Co.
Haviv, A. Q. (2015). MEAN Web Development.
Heitkotter, H., Majchrzak, T. A., Ruland, B., & Weber, T. (2013). Evaluating Frameworks for Creating Mobile Web Apps. Munster, Germany.
Hejlsberg, A. ( 2017). MSBuild Conference.
Hejlsberg, A. (n.d.). MSBuild Conference 2017.
Helm, S. (2015). Evaluation of JavaScript Frameworks. Sweden.
ISO. (2013). Measurement reference model and guide (ISO/IEC 25020:2013). Retrieved from iso: www.iso.org
ISO. (2013). Measurement reference model and guide (ISO/IEC 25020:2013). Retrieved from iso: www.iso.org
ISO. (2019). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Planning and management (ISO/IEC 25001:2019). Retrieved from saiglobal: https://infostore.saiglobal.com/en-us/Standards/DR-AS-ISO-IEC-25001-2019-1147053_SAIG_AS_AS_2719752/
Jaap, K. (2016, June 8). Evaluation of JavaScript frameworks for the development of a web-based user interface for vampires. Amsterdam, Netherlands.
JavaScript Web Applications. (n.d.).
Joanna , D. F., & Laplante, P. A. (2017). A content analysis process for qualitative software engineering research. Innovations in Systems and Software Engineering, 129-141.
jQuery. (2020). Retrieved from jQuery: https://jquery.com/
Krasner, G. E., & Pope, S. (1988). A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk80 System. JOOP - Journal of Object-Oriented Programming , 26-49.
Krause, S. (2018). Results for js web frameworks benchmark . Retrieved from https://stefankrause.net/: https://stefankrause.net/js-frameworks-benchmark8/table.html
Losavio, Chirinos, Matteo, Lévy, & Ramdane-Cherif. (2004). ISO quality standards for measuring architectures. The journal of systems and software, 209-223.
MacCaw, A. (2011). JavaScript Web Applications. Sebastopol: O'Reilly.
Madhuri , J. A., Balkrishna, S. R., & Anushree, D. (2015). Single Page Application using AngularJS. International Journal of Computer Science and Information Technologies, 2876-2879.
Marcos, E., & Lázaro, M. (2006). An Approach to the Integration of Qualitative and Quantitative Research Methods in Software Engineering Research. Philisophiocal Foundations on Information Systems Engineering, 757-767.
Mariano, C. L. (2017, January). Benchmarking JavaScript Frameworks. Dublin.
Miller , J., & Adi, O. (2019, 11 26). Rendering on the Web. Retrieved from developers.google: https://developers.google.com/web/updates/2019/02/rendering-on-the-web
Miller, J., & Osmani, A. (2019). Rendering on the Web. Retrieved from developer google: https://developers.google.com/web/updates/2019/02/rendering-on-the-web
Moradnejad. (2015). Average User Reputation for All Popular Tags. Retrieved from stackExchange: https://data.stackexchange.com/stackoverflow/query/1076963/average-user-reputation-for-all-popular-tags
Mraz, M. (2019). Component Based UI Web Development . Brno.
Napoli, M. (2018). isItMaintained. Retrieved from GithHub Repository: https://github.com/mnapoli/IsItMaintained/blob/master/src/Maintained/Statistics/StatisticsComputer.php
Nitze, A. (2014). Modularity Of JavaScript Libraries and Frameworks in Modern Day WebDevelopment. Berlin.
Nitze, A. (2015). Modularity of Javascript Libraries and Frameworks in Modern Web Applications.

nodejs. (2020, february 28). Retrieved from nodejs: https://nodejs.org/en/
Noring, C., & Deeleman, P. (2017). Learning Angular: A no-nonsense guide to building real-world apps with Angular 5. Packt Publishing.
Overson. (2015). Plato. Retrieved from github: https://github.com/es-analysis/plato
Pekhanova, J. (2009). Evaluating Web Development FrameWorks: Django, Ruby on Rails and CakePHP. The IBIT Report.
Rastogi, S. (2019). framework-vs-library. Retrieved from richwebdeveloper: https://richwebdeveloper.com/framework-vs-library/
ReactJS. (2020). Retrieved from ReactJS: https://reactjs.org/
Robie, J. (2020). What is the document object model. Retrieved from https://www.w3.org/: https://www.w3.org/TR/REC-DOM-Level-1/introduction.html
Savkin, V. (2016). Angular, Why TypeScript? Retrieved from vsavking: https://vsavkin.com/writing-angular-2-in-typescript-1fa77c78d8e8
Schmidt, D. C., & Bushmann , F. (2003). Patterns, Frameworks and Middleware. Their Synergistic Relationship. Proceedings of the 25th International Conference of Software Engineering . Portland: IEEE.
StackOverflow Developer Survey. (2019). Retrieved from StackOverflow : https://insights.stackoverflow.com/survey/2019
Standardization, I. O. (2011). System and software quality models (ISO/IEC 25010:2011). Retrieved from iso: www.iso.org
Standardization, I. O. (2013). Quality Requirements (ISO/IEC 25030:2013). Retrieved from iso: www.iso.org
State Of JavaScript Survey 2018 . (n.d.). Retrieved from Stateofjs: https://2018.stateofjs.com/front-end-frameworks/overview/.
Strimpel, J., & Najim, M. (2016). Building Isomorphic JavaScript Apps. Sebastopol: O'Reilly.
Sun , Y. (2019). Single-Page Applications. In Y. Sun, Practical Application Development with AppRun (pp. 141-162). Berkeley: Apress.
Taivalsaari, A., Mikkonen, T., Cesare , P., & Systä, K. (2018). Client-Side Cornucopia: Comparing the Built-In Application Architecture Models in the Web Browser. WEBIST (pp. 1-24). Seville: LNBIP.
Taivalsaari, A., Mikkonen, T., Systä, K., & Pautasso, C. (2018). Web User Interface Implementation Technologies: an Underview.
Teixeira, L., Xambre, A. R., Alvelos, H., Filipe , N., & Ramos, A. L. (2015). Selecting an Open-Source Framework: A practical case based on software development for sensory analysis. Procedia Computer Science 64, 1057-1064.
The State Of The Octoverse. (2020, January 5). Retrieved from Octoverse: https://octoverse.github.com/
trends built with. (2019). Retrieved from trends.builtwith: https://trends.builtwith.com/javascript/jQuery
TypeScript. (2020). Retrieved from typescriptlang: http://www.typescriptlang.org/
Tyson, M. (2019, 11 25). How to Approach State Management in React, Vue and Angular. Retrieved from Medium: https://medium.com/javascript-in-plain-english/the-art-of-the-state-e19816732530
Vue.js. (2020). Retrieved from vue: https://vuejs.org/v2/guide/
W3C. (2015). fetching-resources. Retrieved from W3C: https://www.w3.org/html/wg/spec/fetching-resources.html
W3C DOM standard. (2020). Retrieved from www.w3.org/TR/REC-DOM-Level-1/level-one-core.html.
Weiringa, R. J. (2016). Design Science Methodology for Information Systems and Software Engineering. Heideberg: Springer.
Wilken, J. (2018). Angular In Action. New York: Manning Publications.
Wilson, E. (2018). MERN quick start guide. Birmingham: Packt Publishing.
Wohlgethan, E. (2018). Supporting Web Development Decisions by Comparing ThreeMajor JavaScript Frameworks: Angular,React and Vue.js. Hamburg, Germany.

# Appendix

1. Function for median resolution time (Napoli, 2018).

```php
    private function computeResolutionTime(array $issues)
    {
        $durations = array_map(function (Issue $issue) {
            return $issue->getOpenedFor()->toSeconds();
        }, $issues);

        return new TimeInterval($this->median($durations));
    }
```

2. Function for percentage open rate (Napoli, 2018).

```php
    private function computeOpenIssueRatio($user, $repository)
    {
        $query = "repo:$user/$repository type:issue " . $this->getExcludedLabelsSearchString();

        $results = $this->github->search()->issues("$query state:open");
        $openCount = $results['total_count'];
        $results = $this->github->search()->issues("$query state:closed");
        $closedCount = $results['total_count'];

        $total = $openCount + $closedCount;

        return ($total !== 0) ? $openCount / $total : 0;
    }
```

3. SQL Query for user reputation per tag Adapted based on original from Moradnejad (Moradnejad, 2015)

```sql
SELECT TagName, CNT, Rep
FROM (
  SELECT TagName, count(*) as CNT, AVG(Cast(rep as Float)) as Rep
    FROM (
      SELECT p.id, T.TagName, u.Reputation as rep
      from posts as p
      inner join PostTags as pt on pt.PostId=p.Id
      inner join Tags as t on t.Id=pt.TagId
      inner join Users as u on u.Id=p.OwnerUserId
    ) as t1
  group by TagName
  ) as t2
WHERE TagName LIKE 'reactjs'
  OR TagName LIKE 'angular'
  OR TagName LIKE 'vue.js%'
```

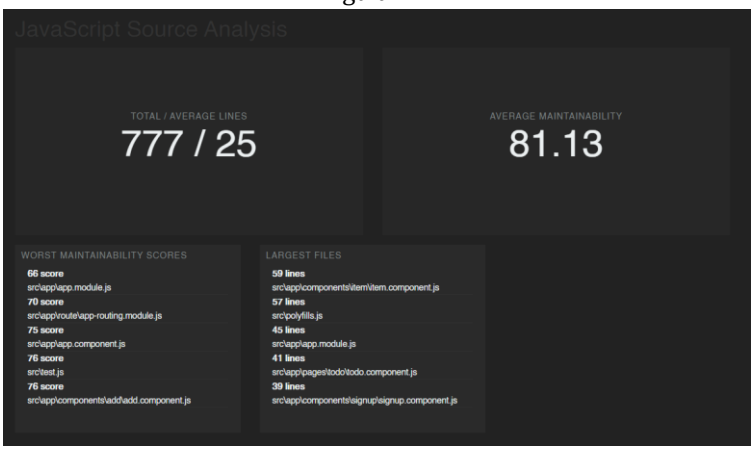
4. SQL Query for question response and quality based on original from Jon Ericson (2017).

```
select TagName,
       isnull(datediff(minute, q.CreationDate, min(a.CreationDate)),
            9999999) TTA
into #answers
from Posts q
     join PostTags on q.Id = PostId
     join Tags t on t.Id = TagId
     left join Posts a ON a.ParentId = q.Id
                          and a.OwnerUserId <> q.OwnerUserId
where q.PostTypeId = 1
      and q.ClosedDate is null
      and q.Score >= 0
      and a.CreationDate > q.CreationDate
      and t.TagName in ('vue.js')
      and q.CreationDate > '2018-01-01'
group by q.Id, q.CreationDate, TagName;

with median_answer as (
  select TagName,
         percentile_disc(0.5)
                 within group (order by TTA)
                 over (partition by TagName)
               median_tta
  from #answers
),

medians as (
  select TagName,
         -- http://stackoverflow.com/a/8846679/1438
         percentile_disc(0.5)
                 within group (order by ViewCount)
                 over (partition by TagName)
               median_views,

         percentile_disc(0.5)
                 within group (order by AnswerCount)
                 over (partition by TagName)
               median_answers,

         percentile_disc(0.5)
                 within group (order by Score)
                 over (partition by TagName)
               median_score
  from Posts p
       join PostTags on p.Id = PostId
       join Tags t on t.Id = TagId
  where PostTypeId = 1
        and TagName in ('vue.js')
),
```

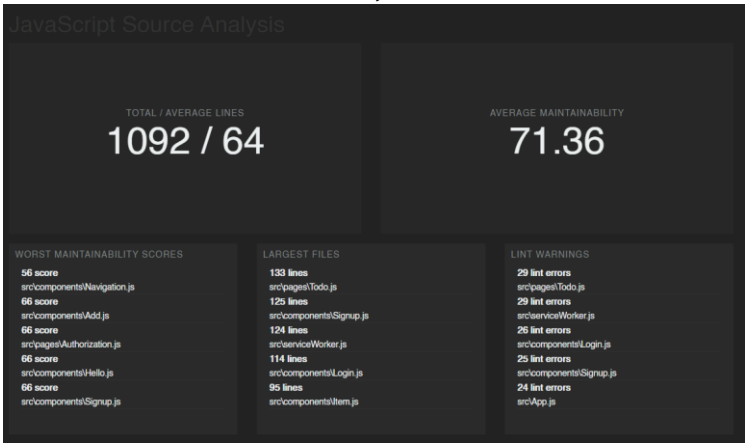
```
averages as (
  select TagName,
         count(*) N,
         avg(ViewCount) avg_views,
         round(avg(1.0*Score), 1) avg_score,
         round(100.0*count(ClosedDate)/count(*), 1) closed_rate,
         round(avg(1.0*AnswerCount), 1) avg_answers,
         round(100.0*count(AcceptedAnswerId)/count(*), 1) accepted_rate,
         round(100.0*count(case
                             when AnswerCount > 0 then 1
                           end)/count(*), 1) answer_rate
  from Posts p
       join PostTags on p.Id = PostId
       join Tags t on t.Id = TagId
  where PostTypeId = 1
        and TagName in ('vue.js')
  group by TagName
)

select m.TagName,
       N,
       median_views,
       avg_score,
       closed_rate,
       avg_answers,
       accepted_rate,
       answer_rate,
       median_tta
from medians m
     join averages c on c.TagName = m.TagName
     join median_answer a on a.TagName = m.TagName
group by m.TagName,
       N,
       median_views,
       avg_score,
       closed_rate,
       avg_answers,
       accepted_rate,
       answer_rate,
       median_tta
order by median_tta
```

5. Plato reports on total and average lines of code and the maintainability index

Angular




React.js




Vue.js