Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Computer Science and Engineering
2021 | LIU-IDA/LITH-EX-A--21/046–SE

An Exploratory Study of Micro Frontends

En Explorativ Studie av Mikrofrontends

Anna Montelius

Supervisor : Chih-Yuan Lin
Examiner : Kristian Sandahl

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

"Microservices" has become a real buzz word in the software development community during the last couple of years. Most developers are familiar with the architectural concept which solves the problem of systems growing to large monoliths too complex to handle. This architecture has however mostly been used in backend development, even though many companies are struggling with large, monolithic frontend codebases. This is where micro frontends come in, an architectural as well as organisational approach to developing applications all the way from presentation to data layer. The micro frontends approach is relatively new, and even though there is some talk about it in the software community, many companies are unfamiliar with it, and there is very limited scientific work performed on the topic. The aim of this study was to investigate strengths of and challenges with micro frontends, and specifically how the modifiability of a web application is affected by developing it as a micro frontends project. The method for fulfilling the aim consisted of several parts. During one part, two frontend prototypes of a web application were implemented, one using a Single Page Application technique and one using a micro frontends technique. Another part consisted of interviewing practitioners in the software field with relevant backgrounds to gain their perspective on micro frontends. The results were also used to evaluate which prototype would be most suitable for the specific web application. During the last part of the method, measurements on the implemented prototypes were performed to be used to estimate the modifiability of the prototypes using a mathematical model of modifiability called SQMMA. Based on the results, this report provides an extensive summary of strengths of micro frontends, among other things that there are both beneficial and disadvantageous aspects of micro frontends when it comes to modifiability, risks that should be considered when adopting micro frontends, and a discussion on when to use it and not.

Acknowledgments

I would like to thank Knightec for giving me the chance to do this thesis work for them, and especially, big thanks to my supervisor at Knightec, Jonathan Olsson, for always being encouraging and supportive. The people from Knightec that participated in the interviews also deserve a huge thanks, it was very interesting talking to you all. Also, a big thanks to my examiner, Kristian Sandahl, who has supported this thesis from the start. Lastly, thank you to my opponent and friend, My Norsbo, who has been supporting me during all these five years.

Contents

List of Figures

List of Tables

1 Introduction

In this chapter, the motivation for the thesis work is presented, giving a brief history of web system architecture and how it has led up to the need for this research. This is followed by the aims of this research. Then, the research questions together with short explanations and the means of answering the questions are presented. Lastly, the delimitations of this work are described.

1.1 Motivation

A lot has happened in the field of web system architecture since the Web was created in the early 1990s [1]. Back then, web pages were delivered by the server to the browser as static documents, and no code could be executed on the client-side. With the evolution of the Web, the complexity of the content increased, and the static pages that web pages originally consisted of were gradually replaced with applications with behaviour similar to desktop applications.

Client-side scripting was introduced in 1995 with the launching of JavaScript, and asynchronous communications between client and server in 2005 with AJAX, which both enabled creation of more dynamic web applications [1]. Since AJAX allows parts of a web page to update without having to reload the entire page, this in a way made the client-side less dependent on the server-side. Sometime after this the concept of Single Page Application (SPA) arose, giving the client-side even more responsibility and complexity. In an SPA, the client-side is responsible for dynamically updating a page with minimal communication with the server-side. A single HTML page with its belonging JavaScript and CSS resources is loaded and after that, the client-side does the frontend related work. In other words, the handling of the presentation layer, which traditionally was the responsibility of the server-side, was moved to the client-side.

The server-side of web applications also became increasingly more complex with the evolution of the Web [1]. With the backends of web applications growing larger, a potential problem faced within many other fields of software than web development arose: large, monolithic systems. Several claims have been made about the limitations of large monolithic applications. Lewis describes in an interview with Thönes [2] that these types of systems become hard to maintain and modify. Richardson [3] also mentions the large risk that they become difficult

to modify, but also difficult to understand and keep modular. Newman [4] claims that even if there is an intention to keep monolithic codebases modular, it becomes hard to stick to that concept as the system grows and as a consequence, the modularity decreases. As an option to the monolithic approach, the microservice architecture came about [5], in which the system is organised into small, independent services that can be deployed separately [4]. The concept has become a real buzzword the last couple of years and has been appropriated by companies such as Amazon, Netflix, and LinkedIn [5]. While this idea has become popular within backend development, many companies are beginning to struggle with large, monolithic frontend codebases on the client-side [6]. Mezzalira's [7] experience tells us that in frontend projects you often start with an SPA and continue with that until the application is so complex that you start to reimplement the application.

Jackson [6], Mezzalira [7], and Geers [8] all suggest the same solution to this problem: micro frontends. The concept of micro frontends can be described as an extension of microservices to the frontend layer [8]. The architecture includes the entire stack, from presentation layer to data storage layer. The concept also introduces an organisational structure that consists of small full stack teams that work independently with a specific microservice operating all the way from the presentation layer to the database layer. According to Geers, micro frontends optimize for feature development and facilitate performing changes in the system, amongst other things. In short, many of the supposed benefits of micro frontends can be summarised as an increase in modifiability.

Although microservices have been around for a couple of years, micro frontends gained traction in 2019 when several organisations started to embrace them [9]. InfoQ listed micro frontends amongst new software architecture trends to watch in 2020, and in their graph showing majority stages of architectural trends, the micro frontends concept was classified in the earliest stage out of four [9]. Even though a few experts on software promise many benefits with micro frontends, how much do we really know about this relatively new concept? How does the micro frontends concept affect the produced software, will the modifiability increase as promised by Geers [8]?

1.2 Aim

The aim of this thesis is to investigate how the software development of web applications is affected by adopting the micro frontends concept, both the architectural and the organisational aspects. The research is of exploratory nature and aims to collect any information of value for evaluating strengths of and challenges with micro frontends. A part of the research is specifically directed at investigating effects of the micro frontends concept on the quality attribute modifiability since many of the supposed benefits of micro frontends can be summarised as an increase in modifiability. The findings of the research will also result in an evaluation of whether it is appropriate to use micro frontends in a specific application presented in 2.2 *Existing Solution*. Two prototypes of the specific application will be implemented using an SPA technique respectively a micro frontends technique. The modifiability of the prototypes will be estimated using a mathematical model of modifiability. Qualitative interviews will be held to collect opinions on the micro frontends concept in general, and the results will amongst other things be used to evaluate the suitability of using micro frontends in the specific application.

1.3 Research Questions

Based on the above described aims of this research, three general research questions were formulated together with more specific sub questions. These are presented in this section.

When the word *practitioners* is used, practitioners in the software field working at the company for which this research is performed are meant.

Because of this research being exploratory, the first research question is wide, and addresses general, possible implications of micro frontends specific concepts. The first sub question deal with possible positive effects of using micro frontends on the development of a web application as well as the software itself. The third addresses what risks should be considered when starting a micro frontends project. The questions will be answered with results from the qualitative interviews.

1. What are the effects of adopting the micro frontends concept in a web application project?
    a) What do practitioners believe are the positive effects on the development of the software as well as the software itself?
    b) What risks should be considered when starting a project using the micro frontends concept according to practitioners?

As described in the motivation above, many of the supposed benefits with micro frontends are related to an increase in modifiability. Because of this, the second general question is targeted specifically at this quality attribute. The first sub question addresses how practitioners think the modifiability of a web application is affected by micro frontends, both when it comes to modifiability of the code and team organisational aspects facilitating modifications in the code. This will be answered with the results from the qualitative interviews. The second question is about if the supposed increase in modifiability in micro frontends is reflected in the frontend code of a micro frontends application and will be answered with the results from the mathematical modelling of the two prototypes.

2. How is the modifiability of an application affected by adopting the micro frontends concept?
    a) What do practitioners believe are the implications of implementing a web application using the micro frontends concept with regards to modifiability?
    b) Can the claim that the modifiability of a micro frontends application is better than that of an SPA be verified with a quantitative modifiability model using code metrics?

The third, and last, general research question addresses when the micro frontends concept should be used. The first sub question asks when the micro frontends concept should be applied in general, and the second question asks if it is suitable in a specific case. All these questions will be answered with the results from the qualitative interviews, and the third additionally with the results from the mathematical modelling.

3. When should the micro frontends concept be used?
    a) In which type of projects do practitioners believe it can be valuable to adopt the micro frontends concept, and when should it not be used?
    b) Is it suitable to use for the application presented in *2.2 Existing Solution*?

1.4 Delimitations

One of the web application prototypes implemented during this thesis will act as an example of how the frontend part of a micro frontends application can be implemented. The quantitative evaluation and comparison of the prototypes will be specific to the implementations. Factors such as implementation techniques used and application size probably impact the results of the quantitative evaluation, and applications using different techniques and of different sizes might yield different results.

The number of interviews conducted for the qualitative evaluation had to be narrowed with respect to the limited time allocated for the thesis. All interviewees are currently employed at the company for which this research is conducted. This could mean that they have company specific experiences that could be reflected in the results. A larger number of interviewees from a various number of companies could yield more representative results.

2 Background

This chapter provides the context of the research work. First, the background of how the research came about and the company that has requested the research is presented. Second, the application that the two prototype applications will be based on is described.

2.1 Origin of Work

This research work is done at the request of Knightec, a consulting firm with expertise in technology, digitalisation, and leadership. Knightec has a web application that is used internally at the company and in which the frontend module is built in an old web framework. This module is in need of a rebuild. It is therefore of Knightecs interest to know if the micro frontends pattern would be suitable to adopt when rebuilding the frontend module, or if it would be better to stick with the traditional monolithic approach.

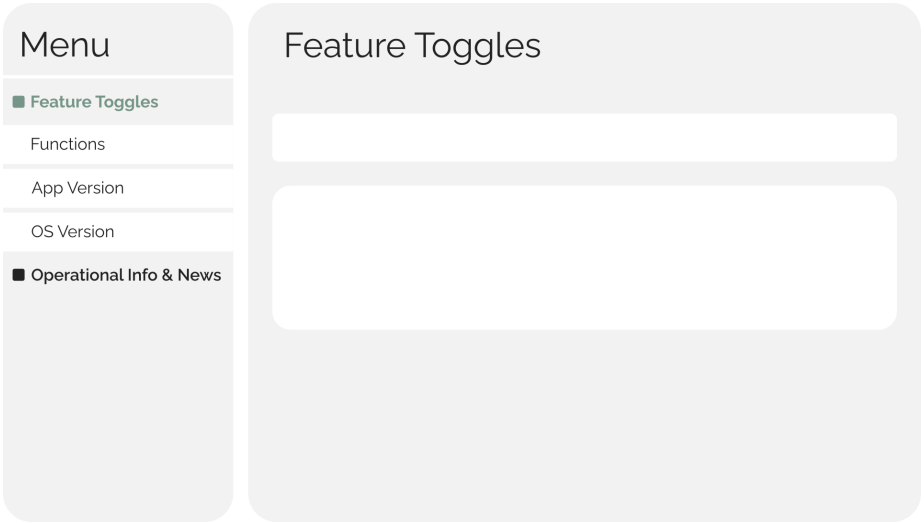

Figure 2.1: Sketch of the UI of the current solution. The menu to the left is used to direct which page is shown in the container to the right.

2.2 Existing Solution

The web application is a support web used for handling other web applications. It is currently implemented in AngularJS [10], an older version of the modern web framework Angular [10]. It is built as an SPA (see *3.1 Single Page Application*) that sits on top of a backend partly consisting of microservices. The application contains a menu that is used for navigating in the application, that is, it controls which page to show in the remaining space of the application. The menu has two sub menus, which are expanded when the title of the sub menu is clicked. One of the sub menus redirects to pages allowing the user to change settings for the applications handled by the support web, and the other sub menu redirects to pages for creating and viewing operational info and news about the handled applications. A sketch of what the UI of the web application looks like is shown in Figure 2.1.

3 Theory

This chapter provides a theoretical base for the research. First, the SPA and micro frontends concept are described. Supposed strengths of, according to the literature, and scientific work on micro frontends are presented and discussed in the micro frontends section. Then, definitions of the quality attributes maintainability, modifiability, modularity, and analysability are provided, since a part of this research is concentrated on modifiability, and these four attributes are tightly related. After that, methods for evaluating software architecture are outlined. Some scientific work using these methods are described. Then, chosen architecture evaluation methods, that is, semi-structured interviews and mathematical models, are presented and discussed, followed by sections about theory needed to understand the mathematical models. Finally, sections on the chosen implementation techniques, React and single-spa, for building the prototypes are included. Concepts of the frameworks that are important for the results of this research are introduced.

3.1 Single Page Application

According to Smith [11], there are two general implementation approaches to building the frontend part of modern web applications today. The first is the traditional approach, which means implementing a lightweight frontend that relies on a server to perform most of the logic, and the second means implementing a Single Page Application (SPA). The traditional implementation style should only be used when the client-side of the application is simple, Smith states, and because the current trend is to have a feature-rich and powerful browser application, the SPA implementation style has gained popularity according to Geers [12].

In an SPA, a single web page is run with the intention of giving the user an experience similar to that of using a desktop application according to [1]. The SPA is the presentation-layer of the application, which can communicate with a backend, but does not rely on the backend to perform all the logic. In contrast to the historical web application implementation style of loading full pages from the server, an SPA does not require full page reloading for updating a small piece of the web page. In Figure 3.1, an architectural diagram of an application with an SPA and a monolithic backend is shown.

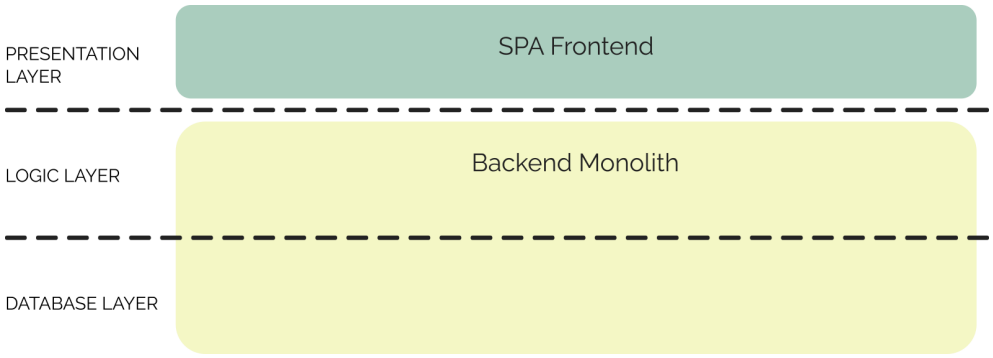

Figure 3.1: Architectural overview of an SPA frontend on top of a monolithic backend.

3.2 Micro Frontends

Geers [12] describes that the current trend in web architecture is to have an SPA as frontend that sits on top of a microservice backend, as shown in Figure 3.2. The idea of micro frontends, which is the topic of this work, is to extend the idea of microservices to include the frontend as well, as described by Geers in his book from 2020 [8]. Further, it is important to understand that the concept is not just an architectural approach for building web applications, but an organisational approach as well. In micro frontends, the presentation layer is divided into small, manageable pieces, as described by Jackson [6], that each provides a specific feature in the web application [8], for example a menu or a video player. A frontend piece only communicates with the backend services that it needs for the feature it handles, which means that a micro frontends system is divided into a number of full stack micro applications [8]. Each micro system is an autonomous application [8] that has its own continuous delivery pipeline that builds, tests, and deploys the micro application [6]. They can even be deployed using different deployment services [6]. As a result, a micro application can function on its own, even if other micro applications are down [8]. Each stack can be built with completely different implementation techniques, even though they together compose a single web application [8]. This type of composed application, that is, a micro frontends application, will from now on in this report be referred to as an MFA.

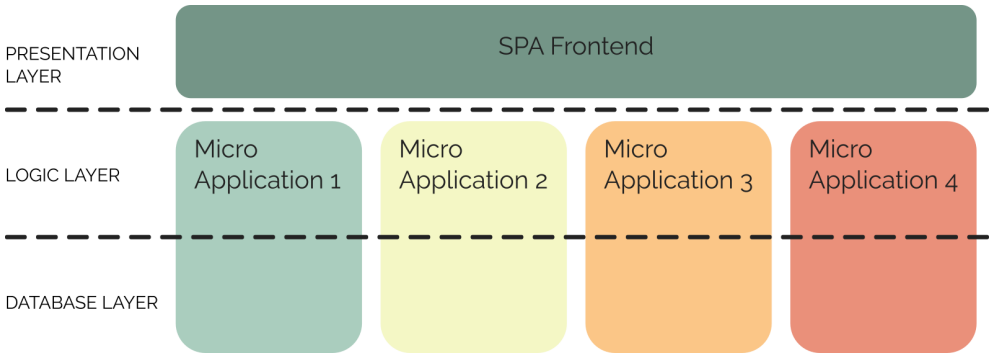

Figure 3.2: Architectural overview of an SPA frontend on top of a microservice backend.

Naturally, since the client- and server-side typically have separate responsibilities and are implemented using different software techniques, the team organisation trend that came with the rise of web development became to have a frontend team and a backend team, as described by Geers [8]. This is known as horizontal teams, he explains. The micro frontends concept,

instead of this, introduces vertical teams that work with a single full stack micro application, from its presentation layer to its data layer. Rather than being grouped by the development technologies being used, vertical teams are grouped by features in the application. An overview of the structure of micro frontends is provided in Figure 3.3.

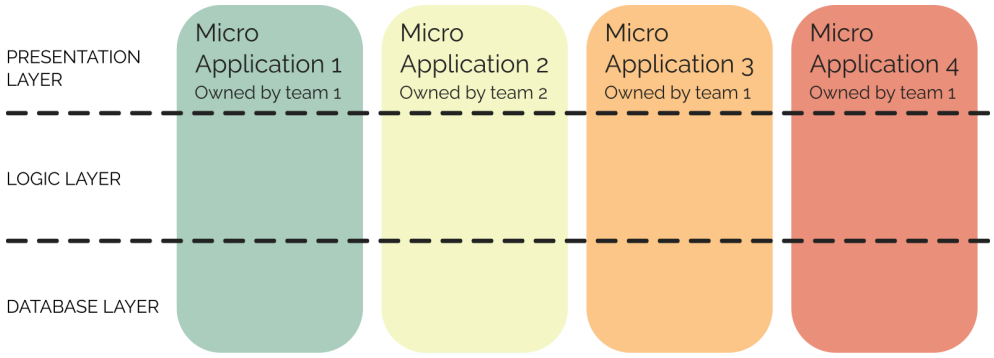

Figure 3.3: Architectural overview of an MFA, where each colour represent a full stack micro application owned by a dedicated team.

3.2.1 Supposed Strengths

According to Geers [8], there are several reasons why companies adopt the micro frontends architecture. One is that it optimises for feature development, since all skills needed to develop a feature is included within a single team. Both frontend and backend developers exist within the same team, which can facilitate the communication needed to develop the new feature compared to if an independent frontend team needs to communicate with an independent backend team.

Maintaining an MFA has its benefits compared to maintaining a frontend monolith, Geers [8] tells us. Since the scope of each micro application is narrow compared to the scope of a monolithic solution, they can be easier to understand compared to trying to understand the entire code base of a monolithic code base. The codebases of the micro applications are smaller than what the codebase of a monolithic solution would be, which can help when refactoring the code. This is also described by Jackson [6], who additionally explains that the small codebases tend to be easier to understand and work with. Further, he mentions that complexity arising from unintentional coupling between modules can be avoided. Myers [13] also suggests that the micro frontends architecture might be simpler than other web architectures, making applications implemented with the architecture easier to reason about and manage.

Another reason for adopting micro frontends that Geers mentions is that it makes frontend upgrades easier [8]. Each full stack application works independently and the team owning it can perform changes on it without interfering with the stack of other teams. Frontend technologies change rapidly, and what was considered the state of the art in frontend development last year might now be considered deprecated. Geers continues with describing that switching from one frontend framework to another in a large monolithic codebase is a lot riskier than doing it for one micro application at a time in a micro frontends codebase. Jackson [6] and Mezzalira [7] mean that organisations often find themselves in a situation where they want to rewrite an entire application because it has become too complex and uses deprecated technology. Micro frontends offer the possibility of doing incremental upgrades to small parts, micro applications, of a system instead of having to upgrade the entire application at once according to Jackson. Myers [13] also describes that the micro frontends architecture can facilitate migration from an old application since the architecture allows for a new application to run side by side with the old one.

Geers [8] also expresses that one idea of micro frontends is to increase customer focus. This, Geers explains, is because each team delivers features directly to the customer instead of delivering through a team dedicated to customer communication.

3.2.2 Scientific Work

Three peer-reviewed research papers on the topic of micro frontends were found. In all three of the research papers, an MFA is implemented. The papers are described and discussed below.

In their research paper from 2019, Yang et al. [14] describe a potential problem with growing SPAs that end up not being well scaled, nor well deployed, and difficult to maintain. Based on this problem analysis, Yang et al. propose the usage of micro frontends in content management systems (CMSs). In their method Yang et al. applied the micro frontends concept in the development of a CMS. Although they refer to Geers [12], only the architectural aspects of micro frontends are mentioned in the report, not the organisational aspects. The CMS was developed using Mooa [15], a micro frontends framework for Angular that is based on single-spa (see *3.9 single-spa*). An architectural diagram is included in the report, and some implementation details. However, no evaluation of the developed system was performed in the study. Not much is said in the report about the qualities of micro frontends in general, except that it is a good idea to apply microservices in the frontend simply because there are benefits to applying them in the backend.

Pavlenko et al. [16] followed a case study of applying the micro frontends concept in an SPA for online courses aggregator service and then analysed advantages and disadvantages of the concept based on the case study. Pavlenko et al. also refer to Geers [12] in their report when describing the principles of micro frontends, but only the architectural aspects are mentioned. Pavlenko et al. present the requirements on the web application, and what architectural decisions were made to fulfil the requirements. An architectural diagram as well as a description of the application is presented, which is followed by an analysis of the micro frontends concept. In one part of the analysis, different approaches to building frontends in web applications are compared, amongst others the SPA approach and the micro frontends approach. Pavlenko et al. claim that micro frontends are hard to set up and maintain because "most of the work should be done manually" [16, p. 61], and specifically, they rank micro frontends as being harder to setup than SPAs. However, it is not clear what Pavlenko et al. are basing the claims on since they are not relating the claims to any reviewed literature or experience gained from the case study. In another part of the analysis, possible challenges with the handling of static assets in an MFA are highlighted. Static assets can for example be images or styles, and Pavlenko et al. describe that the handling of static assets is important to take into consideration when deploying. Since micro applications within the MFA may share static assets, but should be deployed independently according to the micro frontends principle, a solution for giving all separate deployments access to the same static assets needs to be defined, they explain.

In [17] from 2020, Shakil and Zoitl present a work in progress on a micro frontends architecture for industrial Human Machine Interfaces (HMIs), which are UIs that allow a human operator to communicate with a production machine. Shakil and Zoitl argue that when working with a modular machine to which it should be possible to add new modules, the HMI should reflect the changes made in the system. Integration of new HMI objects, corresponding to new machine modules, requires reprogramming and redeploying of the HMI software, they continue, and because this modification should be done with minimal effort, the micro frontends architecture is interesting to consider for HMI systems. When describing micro frontends, Shakil and Zoitl also refer to Geers [12]. However, even though they labelled their research to be about micro frontends and refer to the definition of micro frontends provided in

[12], they too only mention the architectural aspects of the definition. In their method, Shakil and Zoitl developed a micro frontends implementation of a modular HMI consisting of several micro applications and a container SPA. The container PSA composed the final application by using an Import Map to include the micro application. An architectural diagram of the HMI is presented. At the end of the report, Shakil and Zoitl assess the implemented architecture by reasoning about the results of the implementation, but do not address micro frontends specific concepts about their architecture. In the conclusion however, it is described that one of the key features of the developed architecture is that new modules can be integrated in an existing application without having to reprogram the existing application. Altogether, not much is said in [17] about the qualities of micro frontends, except that Shakil and Zoitl think the architectural aspects of the concept could facilitate adding new modules to an existing system, and no assessment performed with scientific methods was included in the work. With that said, the research is described as a work in progress.

3.3 Maintainability

Because a part of this research is concentrated on modifiability, which is tightly related to maintainability according to the ISO/IEC 25010:2011 standard [18], the definition of maintainability presented in the standard is provided in this section. For the same reason, definitions of modifiability as well as the related quality attributes modularity and analysability are also provided.

Software maintainability is described in [18, p. 14] as the "degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers". The modifications can include corrections, improvements, or adaptions of the software as well as the requirements and functional specifications. Maintainability is one out of eight quality attributes composing the product quality model defined in the standard. Each quality attribute consists of a set of sub characteristics, and the sub characteristics of maintainability are modularity, reusability, analysability, modifiability, and testability. The sub characteristics relevant for this work are described below.

3.3.1 Modifiability

Modifiability is defined by the ISO/IEC 25010:2011 standard [18, p. 15] as the "degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality". It also states that modularity and analysability can influence modifiability, and that modifiability is a combination of changeability and stability. Therefore, these attributes are described in the upcoming sections.

3.3.2 Analysability

Analysability is defined by the ISO/IEC 25010:2011 standard [18, p. 15] as the "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 failures, or to identify part to be modified."

3.3.3 Modularity

Modularity is defined by the ISO/IEC 25010:2011 standard [18, p. 14] as the "degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components."

3.3.4 Changeability

Changeability were defined in the ISO/IEC 9126-1:2011 standard [19]. Since that standard is deprecated, the definition of changeability is no longer available through the standard. According to resources referencing to the standard though, such as [20] and [21], the changeability of a system corresponds to how easy it is to perform modification in it.

3.3.5 Stability

Similar to the definition of changeability, the definition of stability provided by the ISO/IEC 9126-1:2011 standard [19] is not available. According to resources using the standard as reference [21], [22], the stability of a system corresponds to the degree to which modifications can be made in the system without introducing unexpected effects such as bugs.

3.4 Architecture Evaluation

Since the aim of this research is to evaluate the micro frontends concept, which includes an architectural approach to implementing web applications, this section presents different approaches for assessing software architecture. Each approach has its own subsection that provides a definition of the approach, but also examples of specific methods and scientific work that uses the methods.

Bosch [23] identifies four approaches for assessing quality attributes of software architecture. These are scenarios, simulation, objective reasoning, and mathematical modelling. Which evaluation method that is best suited to use depends on quality requirements and plausibility of using the method in the specific context, she clarifies. In Del Rosso's work [24] from 2006 for instance, three of the assessment methods were used to evaluate the architecture of an embedded real-time software platform. The methods used were scenario-based architecture assessment, simulation-based evaluation, and experience-based assessment. According to Del Rosso, the different methods are complementary and, when used together, constitute a valuable tool that yields several advantages when evaluating software systems. The four evaluation approaches are scenario-based, simulation-based, experience-based, and mathematical model-based evaluation, and are described in the subsections below.

3.4.1 Scenario-based Evaluation

Scenario-based evaluation is, as the name suggests, based on developing a set of scenarios that concretises the meaning of a requirement, Bosch describes [23]. Scenario-based assessment is particularly useful for, amongst other quality attributes, maintainability, since this attribute can be expressed very naturally through change scenarios according to Bosch. She further explains that one way to evaluate the software maintainability of a system with a scenario-based evaluation could for example be to specify a change profile that includes typical possible changes in the system. The change profile would then be used to estimate the required effort it takes to adapt the system architecture with respect to the changes.

In a study from 2006 [24], Del Rosso used a scenario-based assessment approach that was based on the methods described in [25], [26], and [27], named ATAM, SAAM, and SBAR. The assessment approach was divided into three phases. The first phase was iterative, and consisted of collecting scenarios, classifying them, and assigning them priorities. During the second phase, the architecture evaluation was performed by holding a scenario walk-through meeting with a selection of people. In the third, and last, phase, the results of the assessment were compiled into a report. Del Rosso mentions in the report that it is important that the participants of the second phase possess a good understanding of the architecture under evaluation. Another thing brought up as a lesson learned by applying the

scenario-based assessment is that the scenario collection is one of the most important phases in the assessment. The key to a successful scenario-based assessment is that the assessment participants are part of the process of creating the scenarios, and that they have experience of using this method, Del Rosso explains.

Bosch [23] also emphasises the importance of the scenario-collection, and stresses that for the effectiveness of scenario-based approach to be good, the scenarios have to be carefully chosen. The scenarios have to accurately represent realistic samples to give accurate results.

The architectures created in this thesis research is only implemented by one person. To involve assessment participants in an iterative process of creating scenarios would require the participants to obtain a thorough understanding of implementations they were not at all involved in creating. In addition, and maybe even more important, it would require a lot of the participants' time during more than one occasion. For a scenario-based assessment to yield results of some validity, more than just the developer of the implemented architecture should be involved in the creation of the scenarios. Otherwise, the scenarios could be biased in favour of one of the architectures. For example, in this thesis the modifiability of MFAs compared to of SPAs are investigated since an increase of modifiability is a claimed benefit of micro frontends. Then the scenarios could subconsciously be created to confirm these claims. For all these reasons, performing scenario-based assessments is not a feasible option in this work.

3.4.2 Simulation

The simulation approach to evaluation of architecture is a dynamic analysis method (see section *3.6 Code Analysis and Metrics*), and the idea of it is to perform an assessment on a prototype implementation of a system architecture during execution, as expressed by Bosch [23]. The implementation can contain some fully functional parts, and some simulated ones. Quality attributes are estimated based on the simulation by for example calculating quality assessment metrics for the implementation. For instance, the quality attribute robustness could be assessed by simulating faulty input to the prototype. According to Bosch, the simulation method is particularly useful for evaluating operational quality attributes such as performance, but could be used to estimate maintainability by performing changes in the simulation implementation and measuring the required effort.

The simulation-based evaluation performed in Del Rosso's work [24] had the purpose of assessing the performance of the software. The method included constructing scenarios to be used when measuring performance, but unlike in the scenario-based evaluation, the assessment techniques used were quantitative and not qualitative. The first step of the method was to identify key performance scenarios. During the second step, performance objectives were established, and test plans were prepared. The simulations were conducted during the third step. The last step consisted of evaluating simulation results as well as evaluating improvements, impacts, and trade-offs for the architecture. The entire method was iterative, and did not end until the results were considered to be satisfactory.

As mentioned earlier, Bosch tells us in [23] that simulation-based evaluation first and foremost is suitable for evaluating operational quality attributes such as performance, but that using it for assessing the maintainability of a system is possible. The suggested way of doing this, as mentioned earlier, is to measure the required effort to perform changes in the system. The results from that should also say something about the modifiability of the system, because by the definition provided in *3.3.1 Modifiability*, modifiability represents how easy it is to perform changes in a system. With that said, this method introduces the same dilemma as described in *3.4.1 Scenario-based Evaluation*. Choosing cases for evaluating with how little effort a change can be implemented should be done by more than one person, and the persons

involved reasonably need to have thorough knowledge of the system being assessed. Similar to how both Bosch in [23] and Del Rosso in [24] state that the choice of scenarios in scenario-based assessment affects the results significantly and is of outermost importance, the same reasoning arguably holds for the choice of modification cases. Because of this, performing simulation-based assessments in this research is not to suitable.

3.4.3 Experience-based Reasoning

Objective reasoning based on experience and logical argumentation is the another evaluation approach listed by Bosch [23]. In this approach, the insights of experienced software engineers are the base of the evaluation. The method is useful in exploratory research, when not enough is known about the research subject and it needs to be explored before more specific research is performed, Bosch explains.

In Del Rosso's work [24], the experience-based evaluation was based on semi-structured interviews (see *3.5 Semi-structured Interviews*) with architects, developers, requirements engineers, and all other persons involved in the development activities for the software platform. The first step of the method for conducting an experience-based assessment was to define the scope of the assessment and decide which people to interview. The interviews were held in the second step of the assessment, and the third step consisted of analysing the collected material. In the report, Del Rosso claims that even though the assessment was based on interviews with the stakeholders, its outcome was technical and not subjective. People tend to have subjective answers to technical questions, and because of that, an assessment team cleaned and refined the outcome from subjective beliefs of the interviewees, he further describes.

Because this research is of exploratory nature and experience-based reasoning, as mentioned, is useful in exactly that type of research, this method is of great interest. Conducting semi-structured interviews could not only collect opinions on the implemented architectures of the prototypes, but also data about aspects of micro frontends in general, not just the architectural ones, which is also of great interest in this research.

3.4.4 Mathematical Modelling

Mathematical modelling is an alternative or a complementary to the simulation method and is a static evaluation method (see section *3.6 Code Analysis and Metrics*) according to Bosch [23]. She further describes that research communities have developed mathematical models for evaluating especially operational quality attributes of architectural design models. A variety of code metrics (see *3.6 Code Analysis and Metrics*) have been provided by the software metrics research community to mathematically represent certain quality attributes of software. For example, McCabe's Cyclomatic Complexity metric [28] quantifies the complexity of a system, and it has been shown that there is a correlation between this metric and the maintenance cost of a system, Bosch continues. Therefore, this metric is often used as a part of the assessment method for evaluating the maintainability of a system. Bosch is however clear about the fact that the correlation between a software metric and a software quality attribute is statistical. That a metric generally gives a good prediction of a quality attribute does not mean that it yields an accurate prediction for all systems.

As mentioned by Del Rosso [24], combining different methods of architecture assessment constitutes a powerful evaluation tool. It can be argued that using more than one method gives a more nuanced result. Using a mathematical model in this research to calculate the modifiability of the implemented prototypes together with an experience-based reasoning assessment is feasible with regards to allocated time and resources for this work and would give two different perspectives on the prototypes.

3.5 Semi-structured Interviews

As previously mentioned in *3.4.3 Experience-based Reasoning*, one way to perform an experience-based assessment on an architecture is to hold semi-structured interviews. As Hove and Anda tell us in their report [29] on experiences from semi-structured interviews in software engineering research, qualitative research methods such as semi-structured interviews were originally designed with the purpose of being used in social science. However, they continue, interviews are nowadays often used in empirical software engineering research. Gray [30] describes semi-structured interviews as an interview method that allows the interviewees to explore views and opinions beyond the asked interview questions. In semi-structured interviews, the interviewer has a list of issues and questions to be covered during the interview but may choose not to include them all in all interviews. Further, the interviewer may choose to ask additional questions that may not have been anticipated at the start of the interview. This is different from structured interviews, in which the exact same questions should be asked to all interviewees, and the questions should not be very open-ended. Structured interviews are suitable to use for collecting data for quantitative analysis and is not very different from using a questionnaire. For these reasons, semi-structured interviews are better suited to use than structured interviews when wishing to collect qualitative data, as in this research. In this section, general guidelines for conducting semi-structured interviews are provided together with related work using this method.

Haselböck et al. have performed several studies [31]–[33] on microservice design. In both a Technical Action Research study [31] as well as a literature review [32], Haselböck et al. identified important design areas of microservice design, such as integration, modularisation and fault tolerance. To validate the identified list of areas in the two studies, Haselböck et al. in a later research work [33] conducted ten semi-structured interviews with domain experts. The research questions of the work addressed what important areas of microservice design there are, how important they are and why, and what challenges exist in the areas.

Another software development related study that used semi-structured interviews to answer its research questions was Groher and Weinreich's study [34] about sustainability concerns in software development projects. These regarded subjects such as which factors influence sustainability in software development and how practitioners improve sustainability in their projects.

3.5.1 General Interview Guidelines

In Seamans work [35] about qualitative research methods in empirical studies of software engineering, it is recommended to begin an interview with a short explanation of the research being conducted. Further, Seaman claims that the amount of information being given should be carefully considered. She explains that the interviewee might be less likely to fully participate if they do not understand the goals of the research, but that they on the other hand might filter their responses if they learn too much about the research, resulting in information being left out by the interviewee.

In a qualitative study, all data is potentially useful, and it is therefore important to get the most out of the interviews being held according to Seaman [35]. One approach that may help to accomplish this is to ask questions that cannot be answered with a 'yes' or 'no'. Further, it can be helpful to use an interview guide during an interview, which helps the interviewer to organise the interview. Usually it consists of a collection of open-ended interview questions and different directions the interviewer can take depending on the answers from the interviewee. The data from the interview is optimally collected by recording the interview, since it is usually hard for the interviewer to take notes at the same time as conducting the interview, Seaman points out.

3.5.2 Collection of Demographic Data

When performing research in which data are collected from people, such as performing semi-structured interviews, relevant demographic data on the interviewees should be demonstrated in the report of the research [36]. Therefore, when conducting semi-structured interviews, demographic data need to be collected. In the work of Haselböck et al. [33], this was done as a first part of the interviews. The following questions were asked:

- What is your current affiliation?
- What are your current roles?
- What country are you from?
- How many years of experience do you have in software engineering?
- How many years of experience do you have in dealing with microservices?
- What experience level do you have in dealing with microservices?
- What is the application domain in which you work with microservices?
- From where did you get your microservice knowledge?
- How large is your system, for example, how many microservices do you operate?
- Could you provide some general information about the application of microservices in your context?

To summarise, the demographic questions in the work of Haselböck et al. [33] aimed to obtain information about the current work, the current roles, and experience in software engineering and microservices of the interviewee. Ten practitioners from eight companies in Austria, Germany, and Switzerland were interviewed, where all companies either used microservices or provided consulting on establishing microservices. Most interviewees had architect roles, but also interviewees with other roles such as developers, researchers, employees working with operations or quality assurance, and consultants, were included to collect data from employees with other perspectives than the architects. Even though the number of participants having a specific work role is demonstrated in [33], the numbers indicate that several interviewees have more than one role, but these combinations of roles are not demonstrated in the report. It could be interesting for the reader to know which combinations of roles the interviewees had, since different combinations can indicate that the interviewees have very different technical knowledge. For example, an interviewee working only with operations might not have the same valuable knowledge of microservices as an interviewee working with operations as well as quality assurance.

The work experience within the software field of the interview participants in [33] ranged between two and 29 years with an average of 15.3 years. Specifically within microservices the average working experience of the participants was 2.5 years, ranging from two years to five years. Haselböck et al. further describe that the variety of the participants in terms of their roles, size of the systems they are working with, and application domains shows that the interviewees had broad experience of applying microservices in different application domains and system sizes.

Groher and Weinreich collected demographic data in their study [34] by asking the following questions:

- In which domain does your company operate?
- What is your job title?
- How many years of experience do you have in software development?

- How many years of experience do you have as software architect/lead developer?
- What roles and tasks have you had in your projects?
- What was the size of your projects (number of developers, project duration)?

Ten team leaders or lead developers from nine companies in Austria were interviewed in [34], and the companies were selected based on the authors' contacts. All interviewees were men with at least a master's degree in software engineering, and with an average work experience within software engineering of more than twelve years, the range being four to 22 years. The number of years of the interviewees working as software architects or lead developers ranged from two to ten years, with an average of six years.

3.5.3 Interview Questions

Which questions to include in the interview guide is dependent on the research work. In the work of Haselböck et al. [33], the second part of the interviews had the purpose of collecting data to be able to answer the research questions. It consisted of both open-ended as well as specific questions regarding microservice design areas. The following questions were asked:

- How would you assess the importance of the listed design areas for a microservice architecture? Please rate on a scale of 1 to 5, where 1 is not important and 5 is very important.
- What is the rationale for your assessment of each design area?
- What are the challenges in the area?
- Is the list missing any important design area?

The first question was a closed question, but the last three were open-ended to allow the interviewee to ask questions to the interviewers and openly discuss the logic behind their answers [33]. When the open-ended questions were being answered, the interviewers asked questions that were not predefined and included in the interview guide.

Similar to Haselböck et al., Groher and Weinreich also included both open-ended and specific questions to not only collect foreseen information but also unexpected information in their study [34]. They further describe that they thought they would obtain more specific information on how practitioners think by conducting the interviews rather in the style of discussions than formals interviews following strict rules. An interview guide that was divided into three parts was used, the first addressing the demographic data as described in *3.5.2 Collection of Demographic Data*. The second part consisted of open-ended questions about the research subject, and the third of closed. The following questions are examples of questions included in the second part:

- General questions about sustainability
  - What is sustainability in software development from your point of view?
  - Is sustainability an important quality in your working context and why?
- Specific questions about sustainability
  - What are the changing conditions that possibly influence sustainability in your software development projects?
  - Which resources need to be handled efficiently and why?

An example of a question included in the third part of the interview guide in Groher and Weinreich's work [34] is the following:

- How important are the following factors with respect to sustainability (1 to 5 scale, 5 means very important)?
  a. Deadlines
  b. Budget
  c. Personnel
  d. Organizational structure
  e. Business infrastructure
  f. Functional requirements
  g. Non-functional requirements
  h. Technologies
  i. Company guidelines and standards
  j. Business goals

3.5.4 Data Processing

After an interview, the data that was collected during it need to be analysed. Both Haselböck et al. [33] and Groher and Weinreich [34] followed the data analysis procedure suggested by Mayring [37] for analysing the data collected during their interviews. That procedure consists of four steps, and how Haselböck et al. interpreted how to use these in their work are described below.

Step 1: Data preparation
During the data preparation, the recorded interviews were transcribed, excluding answers to the demographic questions, which were directly recorded in a table instead.

Step 2: Definition and revision of coding categories
The second step only included data collected during the second part of the interviews and consisted of defining main data categories. A deductive procedure was used, meaning that the coding categories are defined before codifying the data. This resulted in four main categories which were importance, rationale, challenges, and additional areas. As suggested by Mayring [37], the categories were assigned definitions, examples, and coding rules.

Step 3: Data codification
As in the second step, only data collected during the second part of the interviews were processed during the third step. The transcriptions were read and text fragments from them were assigned to one or more main categories defined in the second step. Subcategories were added to the categories as the transcriptions were read if the person reading thought it necessary. After adding subcategories, already encoded fragments were analysed again to determine if they should be encoded as the newly defined subcategory.

Step 4: Interpretation of results
During the last step, an average value for each numeric demographic data was calculated, such as number of years of working experience in software engineering. For data on the roles of the interviewees, the numbers of mentions of each role were summed, and for country and application domains of participants, a list was made. For the numerical data collected during the second part of the interviews, a table that included the data was created. To interpret the non-numerical data, similar answers to questions were connected and summarised, and all answers were collected as results of the study.

3.5.5 Threats to Validity in Semi-structured interviews

A lot of factors can influence the validity of the results from semi-structured interviews. Haselböck et al. [33] address threats to validity in their discussion. They specifically discuss that interviewees with varying roles from different countries, different companies of varying sizes, and working with different system sizes were carefully selected to obtain opinions from people with different backgrounds and perspectives on microservices. They do however explain that they are aware of the fact that the small sample size of interviewees limits the generalisability of the results. Additionally, they claim that the risk of interviewer bias was mitigated by having the interviews being conducted by two interviewers, one who moderated the interview and one who had the chance to ask asked specific questions. Further, Haselböck et al. describe that the analysis of the collected data is a potential threat to validity, but that they mitigated this threat by transcribing all interviews word for word and defining the data categories before reading the transcripts. For the reader, it is not entirely clear why this would mitigate the risk, but it can be speculated that Haselböck et al. mean that transcribing the interviews word for word before summarising the content of the interviews could result in more accurate summaries than if creating summaries by listening to the interviews. Haselböck et al. also address the potential threat that the results do not really follow from the collected data, and that this threat was mitigated by using their method for defining the main categories and subcategories. Lastly, they mitigated the risk that the study would lead to different results if repeated by other researchers by letting two researchers perform all method steps instead of one.

Groher and Weinreich [34] also discuss the validity of their study, and mention many of the threats as well as attempted mitigations of threats that Haselböck et al. [33] do. Like Haselböck et al., Groher and Weinreich explain that they are aware of the threat to the generalisability of the results as a consequence of the limited number of samples, but that they mitigated this risk by selecting interviewees with a high number of years of experience within software development and from different domains and companies. They also let both authors of the study to perform interviews and used an interview guide to reduce interviewer bias, and transcribed the interviews word for word before letting both authors perform the codification of the data, also to reduce the threat to reliability.

3.6 Code Analysis and Metrics

Because the two concepts *dynamic code analysis* respectively *static code analysis* are used in this report, these concepts are described in this section. They constitute two categories of code analysis methods that can be used when assessing software. Dynamic code analysis is defined in the ISO/IEC/IEEE 24765 standard [38, p. 149] as the "technique that relies on observation of system or component behaviour during execution, without need for post-execution analysis, to detect errors, violations of development standards, and other problems". In other words, the code is analysed during execution. The standard defines static code analysis as the contrast to dynamic code analysis, and describes it as the "process of evaluating a system or component based on its form, structure, content, or documentation" [38, p. 439]. That is, static analysis is performed on the code itself, not during execution. Both dynamic and static code analysis generally result in derived code metrics. A code metric, also sometimes called a software metric, is the mapping of a code characteristic to a numerical value [39]. As mentioned in *3.4.4 Mathematical Modelling*, a code metric can be used for mathematical modelling a system with respect to a certain software quality attribute. Below, the code metrics mentioned in this report are provided together with the definitions of them that are used in this work.

3.6.1 Average Method Size (AMS)

In this report, this metric is referred to as *AverageMethodSize* and is defined as the average lines of logical code in all methods. With logical lines of code, lines of code that are not comments are meant. That is, if M is the set of methods defined in a system, and *LLOC* is the logical lines of code of a method, then the *AverageMethodSize* is defined by Equation 3.1.

$$AverageMethodSize = \frac{1}{|M|} \sum_{m \in M} LLOC_m \tag{3.1}$$

3.6.2 Cohesion Among Methods of Class (CAM)

Bansiya and G. Davis presented a model for object-oriented design quality assessment called QMOOD in [40] that defined a number of design properties and mapped them to a number of design metrics. One of the metric was Cohesion Among Methods of Class (CAM), which they describe "computes the relatedness among methods of a class based upon the parameter list of the methods" [40, p. 8]. The metric was actually first defined in [41].

Let T be the union of all object types in the parameters of the methods of a class with n methods, M_i be the set of parameter object types for a method i of the class, and P_i be the intersection of M_i with T for a method i of the class. Then, according to [41], the CAM is defined as in Equation 3.2.

$$CAM = \frac{1}{|T| \times n} \sum_{i=1}^{n} |P_i| \tag{3.2}$$

3.6.3 Coupling Between Object Classes (CBO)

In this work, the metric Coupling Between Object classes (CBO) is used as it is defined in [42] by Gyimothy et al. The ISO/IEC/IEEE 24765 standard provides several definitions of coupling, one of which is "in software design, a measure of the interdependence among modules in a computer program" [38, p. 107]. Gyimothy et al. tell us that a class is coupled to another class if it uses its member functions or instance variables, and therefore describes that the CBO metric "gives the number of classes to which a given class is coupled" [42, p. 898]. In this report, it is referred to as *CouplingBetweenObjects*. Coupling is however usually divided into afferent and efferent coupling. The afferent coupling of a class corresponds to how many other classes uses members of that class, and efferent coupling of a class corresponds to how many classes that class is using members of [43].

3.6.4 Data Access Metric (DAM)

The QMOOD model also defined the metric Data Access Metric (DAM), which is defined as "the ratio of the number of private (protected) attributes to the total number of attributes declared in the class" [40, p. 5].

3.6.5 Design Size in Classes (DSC)

Design Size in Classes (DSC) is also included in QMOOD, and is defined as "a count of the total number of classes in the design" [40, p. 5].

3.6.6 Depth of Inheritance Tree (DIT)

The Depth of Inheritance Tree (DIT) metric comes from Chidamber and Kemerers metric suite for object oriented design presented in [44], and is simply the number of ancestors of a class. In this report, it is referred to as *DepthOfInheritance*.

3.6.7 Lack of Cohesion on Methods (LCOM)

The Lack of Cohesion on Methods (LCOM) metric is another metric presented in [44] by Chidamber and Kemerer. One definition of cohesion provided by the ISO/IEC/IEEE 24765 standard is "in software design, a measure of the strength of association of the elements within a module" [38, p. 74]. Gyimothy et al. describe LCOM as the "number of pairs of member functions without shared instance variables, minus the pairs of member functions with shared instance variables" [42, p. 898]. However, they continue, if the result is negative, the metric is set to zero [42]. In this report, it is referred to as *LackOfCohesion*.

Let $\{I_j\}$ be the set of instance variables used by a method M_i of a class, $P = \{(I_i, I_j) \mid I_i \cap I_j = \varnothing\}$, and $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \varnothing\}$. Then, the *LackOfCohesion* is defined as in Equation 3.3 according to [44].

$$\begin{aligned} LackOfCohesion &= |P| - |Q| \text{ if } |P| > |Q| \\ &= 0 \text{ otherwise} \end{aligned} \tag{3.3}$$

3.6.8 Method Inheritance Factor (MIF)

Brito e Abreu and Carapuça presented a metrics set called MOOD in [45]. One of the metrics included in the set was the Method Inheritance Factor (MIF) [45], which Harrison et al. describe in [46, p. 232] as the "count of the number of inherited methods as a ratio of total methods". In this report, it is referred to as *MethodInheritanceFactor*.

3.6.9 Number Of Children (NOC)

Another metric defined in [44] by Chidamber and Kemerer is Number Of Children (NOC), which corresponds to the number of direct descendants for a class. In this report, it is referred to as *NumberOfChildren*.

3.6.10 Number of Hierarchies (NOH)

Number of Hierarchies (NOH) is a QMOOD metric defined in [40, p. 5] as "a count of the number of class hierarchies in the design.

3.6.11 Number of Methods (NOM)

Number of Methods (NOM) is yet another QMOOD metric defined in [40, p. 5] as "a count of all the methods defined in a class".

3.6.12 Nested Block Depth (NBD)

Oliviera et al. describes the metric Nested Block Depth (NBD) as the depth of the nesting in a code block [47]. In Listing B.4 for example, the maximum NBD of function `create_grid` is two. In this report, the NBD is referred to as *NestedBlockDepth*.

```
def create_grid(width, height):
    grid = [width][height]

    # Loop through grid and set each element to 0.
    for col in range(width):
        for row in range(height):
            grid[col][row] = 0

    return grid
```

Listing 3.1: An example of a Python function in which the maximum NBD is two.

3.6.13 Polymorphism Factor (POF)

The metrics set MOOD also includes the metric Polymorphism Factor (POF) [45], which according to Harrison et al. is defined as the "number of overriding methods in a class as a ratio of the total possible number of overridden methods" [46, p. 232]. In this report, it is referred to as *PolymorphismFactor*.

3.6.14 Response For a Class (RFC)

Chidamber and Kemerer [44] also defines a metric called Response For a Class (RFC). The response set, $\{RS\}$, of a class is a "set of methods that can potentially be executed in response to a message received by an object of that class" [44, p. 487]. The RFC for a class is defined as the number of elements in the response set [44]. In this report, it is referred to as *ResponseForClass*.

Let M be the set of methods of a class, and R_i the set of methods called by method i of the class. Then RS for that class is given by Equation 3.4, and *ResponseForClass* is given by Equation 3.5 according to [44].

$$RS = M \cup_{all\ i} R_i \tag{3.4}$$

$$ResponseForClass = |RS| \tag{3.5}$$

3.7 Mathematical Models of Modifiability

As described in *3.4.4 Mathematical Modelling*, one way to estimate a specific quality attribute of a software system is to model it mathematically with software metrics related to the quality attribute. Since one part of this work is to investigate how the quality attribute modifiability is affected by using the micro frontends architectural pattern, models of modifiability have been sought. Two software quality models that provide different mathematical definitions of modifiability have been found. One model is called SQMMA and is defined by Chawla and Chhabra in [48], and one model is defined by Dayanandan and Kalimuthu in [49]. Both are based on software metrics for object-oriented code, and are described in detail in *3.7.1 SQMMA* respectively *3.7.2 Model by Dayanandan and Kalimuthu* below.

3.7.1 SQMMA

Chawla and Chhabra propose a quality model for assessing the maintainability of software called Software Quality Model for Maintainability Analysis (SQMMA) in their work from 2015 [48]. They describe that even though there exist many models for quality analysis, they mainly provide guidelines on how to assess quality on an abstract level and offer little support on how to assess it in practice. The paper originally provided a way to mathematically model the maintainability of a system as maintainability was defined in ISO/IEC 9126-1:2001 [19], where modifiability was not included. Each quality attribute that maintainability was defined to consist of in ISO/IEC 9126-1:2001 [19], that is, stability, changeability, testability, and analysability, was modelled as a function of certain code metrics. Later, Chawla and Chhabra updated the report to include modifiability as well, and how they did that will be explained later. First, the method of the original paper will be described.

To construct the models, Chawla and Chhabra [48] first mapped design metrics with the chosen quality attributes. The mapping was based on the work of Samoladas et al. [50], who defined a maintainability model for code written in the programming language C. Because of this, some of the design metrics were replaced or adjusted in [48] to not be based on C specific properties. Changeability and stability, two of the quality attributes included in the model, were defined as functions of the design metrics listed in Table 3.1.

Chawla and Chhabra [48] then concretised the design metrics by mapping them to specific software metrics provided by several metrics calculation tools. These software metrics are in [48] referred to as the abbreviations listed in Table 3.1.

Quality Attribute	**Design Metric**	**Metric Name**
Changeability	Average size of statements	TLOC
	Coupling between objects	CBO
	Depth of inheritance tree	DIT
	Lack of cohesion	LCOM
	Method inheritance factor	MFA
	Number of nested levels	NBD
	Polymorphism Factor	NORM/NOM
Stability	Coupling between objects	CBO
	Depth of inheritance tree	DIT
	Directly called components	RFC
	Number of children	NSC
	Number of entry/exit points	1

Table 3.1: Mapping between quality attributes and design metrics together with corresponding practical metric names as defined in the SQMMA model presented in Chawla and Chhabra's work.

For the reader, the definitions of some of the metrics mapped to changeability and stability in [48] are not clear. Chawla and Chhabra do not provide explicit definitions of neither the design metrics nor the specific software metrics. Because of this, definitions that are generally accepted and used in the software field are adopted in this report. How the assumptions about which general definitions to use were made is described in the following paragraph.

Some of the design metrics listed in [48] have a general definition in the software metric world and are in [48] mapped to a code metric that is usually associated with that design metric. These are *coupling between objects*, *lack of cohesion*, and *depth of inheritance tree*. Their generally accepted definitions are provided in *3.6.3 Coupling Between Object Classes (CBO)*, *3.6.7 Lack of Cohesion on Methods (LCOM)*, and *3.6.6 Depth of Inheritance Tree (DIT)*. Other design metrics are generally called a different name in the software field than what Chawla and Chhabra call them, and therefore the more generally accepted name will be used in this report. These are *number of nested levels*, which will be called *nested block depth*, and *directly called components* which will be called *response for a class*. The generally accepted definitions of these are provided in *3.6.12 Nested Block Depth (NBD)* and *3.6.14 Response For a Class (RFC)*. The design metrics referred to as *polymorphism factor*, *method inheritance factor*, respectively *number of children*, have in [48] been mapped to code metrics called NORM/NOM, MFA, respectively NSC. Since these are not the code metrics usually associated with these design metrics, and Chawla and Chhabra do not provide any definitions of the code metrics they have used, the definitions found in *3.6.13 Polymorphism Factor (POF)*, *3.6.8 Method Inheritance Factor (MIF)*, and *3.6.9 Number Of Children (NOC)* are used in this report, which correspond to generally accepted definitions in the software field. No general definition of the design metrics *average size of statements* or *number of entry/exit points* that are referred to in [48] have been found, and therefore assumptions had to be made about the definitions of these. The metric name TLOC, which the design metric *average size of statements* has been mapped to, stands for Total Lines Of Code, which does not align with the name of the design metric. The definition used in this report as a substitute for this unknown metric can be found in *3.6.1 Average Method Size (AMS)*. In [48], *number of entry/exit points* is set to the value one, and therefore the value is set to that in this work as well. A mapping between named design metric in [48] and what definition is used in this report can be found in Table 3.2.

Metric in SQMMA	**Metric in this report**
Average size of statements	*3.6.1 Average Method Size (AMS)*
Coupling between objects	*3.6.3 Coupling Between Object Classes (CBO)*
Depth of inheritance tree	*3.6.6 Depth of Inheritance Tree (DIT)*
Directly called components	*3.6.14 Response For a Class (RFC)*
Lack of cohesion	*3.6.7 Lack of Cohesion on Methods (LCOM)*
Method inheritance factor	*3.6.8 Method Inheritance Factor (MIF)*
Number of children	*3.6.9 Number Of Children (NOC)*
Number of entry/exit points	*No definition*
Number of nested levels	*3.6.12 Nested Block Depth (NBD)*
Polymorphism factor	*3.6.13 Polymorphism Factor (POF)*

Table 3.2: Mapping between metrics mentioned in Chawla and Chhabra's report and what definitions are used for them in this report.

Once Chawla and Chhabra [48] had chosen which code metrics each quality attribute would be a function of, a web-based questionnaire was designed to send out to experts. The purpose of the questionnaire was to collect opinions on how to prioritise each code metric when establishing weights for them. 71 responses from professionals within the software field were collected. The weights were then determined using the AHP technique (see [51] for details on the AHP method). If using the metric names in the right column in Table 3.2, the resulting

functions for calculating changeability and stability as defined by Chawla and Chhabra can be found in Equation 3.6 respectively Equation 3.7 below.

$$\begin{aligned} Changeability = &- 0.13 * AverageMethodSize \\ &- 0.16 * CouplingBetweenObjects \\ &- 0.15 * DepthOfInheritance \\ &- 0.13 * LackOfCohesion \\ &- 0.14 * MethodInheritanceFactor \\ &- 0.16 * NestedBlockDepth \\ &- 0.13 * PolymorphismFactor \end{aligned} \tag{3.6}$$

$$\begin{aligned} Stability = &- 0.20 * DepthOfInheritance \\ &- 0.21 * NestedBlockDepth \\ &- 0.19 * NumberOfChildren \\ &- 0.21 * ResponseForClass \\ &- 0.18 * 1 \end{aligned} \tag{3.7}$$

To then define maintainability as a function of the quality attributes, the same techniques as used for the weights of the code metrics was used by Chawla and Chabra [48] to define the weights of the quality attributes. Although modifiability was not included in the first version of Chawla and Chhabra's work, the revised version provides a model for maintainability as defined in ISO/IEC 25010:2011 [18] where modifiability is included. Because modifiability is said to be a combination of changeability and stability in ISO/IEC 25010:2011, Chawla and Chhabra defined it as a function of these two quality attributes in the revised version of their report. The weights of changeability and stability were also calculated using the AHP technique. Chawla and Chhabra's function for calculating modifiability can be found in Equation 3.8 below, where *Changeability* and *Stability* are defined as in equation Equation 3.6 respectively Equation 3.7.

$$Modifiability = 0.42 * Changeability + 0.58 * Stability \tag{3.8}$$

Chawla and Chhabra in [48] also tested applying their model on four versions of the same source code. In their report, the calculated results are presented as normalised values. The normalisation was performed by dividing the result of each version with the result of one of the source code versions. The modifiability score of the source code versions are presented as normalised values in Table 3.3.

That Chawla and Chhabra in explores a solution for mapping quantifiable code metrics to the high level quality attribute maintainability is interesting, since they claim in [48] that there exist little to no practical guidelines for doing so. However, the fact that no explicit definitions of the design metrics are provided in [48] raises questions about the reproducibility of the report. The readers should not have to search for definitions themselves, especially when it comes to the metrics that do not have generally accepted definitions. The purpose of the work was to provide mathematical functions based on code metrics to model high level quality attributes, and that naturally should mean that the model should be easy to use.

Source code version	Modifiability
7.0.6	1.00
7.0.22	1.02
7.0.39	1.01
7.0.47	1.04

Table 3.3: Normalised values of the modifiability score of the source code versions in Chawla and Chhabra's work.

3.7.2 Model by Dayanandan and Kalimuthu

Dayanandan and Kalimuthu defined a mathematical maintainability model in their study from 2018 [49], which they claimed to be superior to the SQMMA model according to the results of their analysis. Maintainability was defined as a function of the quality attributes that maintainability consists of according to the ISO/IEC 25010:2011 standard [18], there amongst modifiability, and the quality attributes in turn were defined as functions of certain selected software metrics. These came from the metric set Quality Model for Object Oriented Design (QMOOD) [40]. To decide which code metrics the quality attributes should be mapped to, Dayanandan and Kalimuthu sent out a web-based questionnaire to experts to obtain data on how important certain code metrics are for a certain quality attribute. As a result, the software metrics mapped to modifiability are listed in Table 3.4 together with which design property they represent. The definitions of these software metrics can be found in *3.6.2 Cohesion Among Methods of Class (CAM)*, *3.6.4 Data Access Metric (DAM)*, *3.6.5 Design Size in Classes (DSC)*, *3.6.10 Number of Hierarchies (NOH)*, and *3.6.11 Number of Methods (NOM)*.

Design Property	Metric
Cohesion	Cohesion Among Methods of Class (CAM)
Encapsulation	Data Access Metrics (DAM)
Design size	Design Size in Class (DSC)
Hierarchies	Number of hierarchies (NOH)
Complexity	Number of Methods (NOM)

Table 3.4: Design properties used for modelling modifiability in Dayanandan and Kalimuthu's work together with corresponding software metrics.

To establish weights to use for the code metrics in each quality attribute function, the FAHP with Buckley method (see [49, p.90–91] for details on the FAHP method) was used [48]. Given the provided definitions of the code metrics, Dayanandan and Kalimuthu defines modifiability as in Equation 3.9 below.

$$\begin{aligned} Modifiability = &+ 0.60 * Hierarchies \\ &+ 0.65 * Cohesion \\ &+ 0.62 * Complexity \\ &- 0.42 * Encapsulation \\ &- 0.45 * DesignSize \end{aligned} \tag{3.9}$$

To evaluate their defined model of maintainability, Dayanandan and Kalimuthu [49] performed an experimental analysis in which they compared their method with the SQMMA method presented in *3.7.1 SQMMA* and what they refer to as "AHP". All three methods were used on five versions of one code base, and four versions of another. It is however not made clear to the reader what they mean by AHP and how that method differs from the SQMMA method, since SQMMA uses AHP. The maintainability index (MI), which was first designed by Oman and Hagemeister [52] in 1992, was used to validate the results from the different models by calculating the MI for all code bases and using that value as a reference for what was assumed to be the truth about the maintainability of the code bases. The results from the evaluation showed that the model designed by Dayanandan and Kalimuthu was far more accurate on all code bases than the SQMMA and AHP method. However, no discussion on the validity of these results is presented, and the conclusion made by Dayanandan and Kalimuthu, that is, that their model is more accurate than the SQMMA model, should probably be taken lightly since it is of their interest to depict their model as being the best. Several resources, such as [53] and [54], point out concerns about using MI as a measure of maintainability. It could be argued that using MI without discussing these concerns can be seen as a threat to the validity of the results presented in [49].

3.8 React

The tool used when implementing the prototypes in this thesis, React [55], is a JavaScript library for building UIs developed by Facebook. It is component-based, meaning that the frontend is constructed by small, encapsulated components that manage their own states [55]. A component could for example be a button or a text label. In a survey from 2019 with 21 717 participants distributed all over the world [56], React had, out of React, Vue.js, Angular, Preact, Ember, and Svelte, the highest percentage of people having used the framework before, and wanting to use it again [57]. Below, some important aspects of React concepts are described.

3.8.1 React Components

Components in React can be implemented either as classes or functions, which are equivalent from React's point of view [58]. They accept arbitrary inputs, which are called "props", short for properties [58]. Both class and function components can have states, but states in function components were not introduced until the release of React 16.8, which came with an addition called hooks [59]. The reason why hooks were added to enable the usage of states in function components was that the developers found that class components confuse people and introduce issues for today's tools [59]. Because of these reasons, it was decided to use functional components when implementing the prototypes in this work.

3.8.2 Composition

The developers of React recommend that composition is used instead of inheritance when implementing React components [60]. Just like for inheritance in object-oriented code, composition is applied to reuse code between components [60]. It can be used in the form of containment, or in the form of specialisation [60]. Containment simply means that a component contains sub components, which are passed to the component with a special prop called "children" [60]. Specialisation means that a component is a "special case" of another component, which means that the special case component renders the more generic component and configures it with props [60], similar to how sub classes in object-oriented code are special cases of super classes.

3.9 single-spa

The JavaScript framework single-spa [61] is mentioned in [8] as an option for implementing micro frontends. Because this was the only framework found that allowed the micro applications to be implemented with different frontend frameworks, and this described as a key feature of micro frontends, it was chosen to be used for the MFA prototype. A single-spa application consists of a root application and a number of micro applications. The root application renders an HTML page and a JavaScript file that registers micro applications. Each micro application is registered with a name, a function to load the code of the micro application, and a function that determines when the micro application should be active. The micro applications must know how to mount and unmount itself from the DOM, but do not have their own HTML page.

3.10 Exploratory Testing

Because the phrase *exploratory testing* is used in the method of this paper, an explanation of it is provided in this section. Exploratory testing was coined by Cem Kaner [62], who describes it as "a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the value of her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project" [62, p. 36]. In practice, this means that test cases are not defined in advance but rather invented by the tester simultaneously with the test execution. The tester tests the system by exploring it and is steered only by their own creativity and curiosity.

4 Method

In this chapter, the method of the research is presented. First, an overview of the structure of the conducted method is described. This is followed by sections describing the different parts of the method in more detail.

4.1 Overview

The method was divided into four phases. The first phase, the pre-study, consisted of gathering relevant theoretical knowledge for the research work. During the second phase, the implementation phase, two prototypes of the web application described in *2.2 Existing Solution* were developed with the purpose of evaluating which implemented architecture was most suitable to use in the application. Further, an analysis program for deriving code metrics that would be used later was implemented, as well as unit test for confirming that the analysis program worked as intended. The third phase, the data collection phase, included both semi-structured interviews with practitioners in the software field and measurements on the developed prototypes. The purpose of the phase was to collect data that would be used to answer the research questions. The fourth phase, the data processing phase, aimed to process the collected data to be able to analyse them and use them to answer the research questions. Details of the pre-study phase and the implementation phase are described in *4.2 Pre-study* and in *4.3 Implementation*. The activities performed during the data collection phase are described both in *4.4.1 Data Collection* and *4.5.1 Data Collection*, while the activities performed during the data processing phase are described in *4.4.2 Data Processing* and *4.5.2 Data Processing*. A graphical overview of the method is shown in Figure 4.1.

4.2 Pre-study

In the beginning of the work, a pre-study consisting of a literature study was performed to obtain theoretical knowledge about the research subjects. Google Scholar was used to search for scientific papers and Google was used to search for other resources such as blogs, newspapers, and technical standards. Search words related to each subject of relevance were used. When relevant resources were found, the references of these resources were also considered to review.

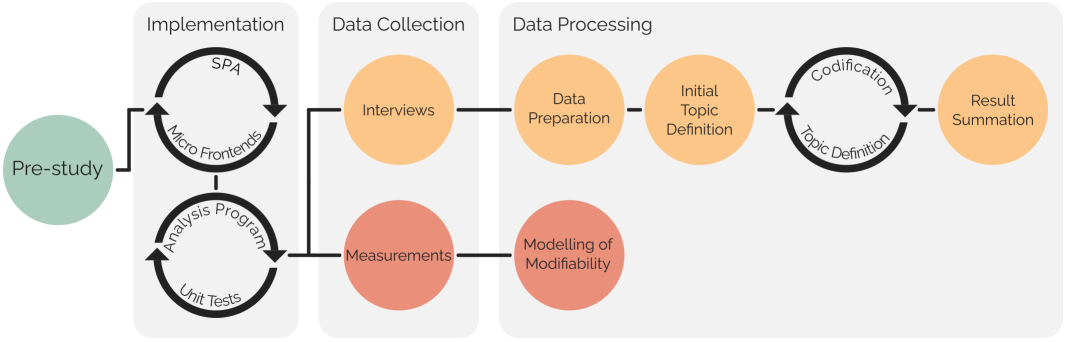

Figure 4.1: Graphical overview of the research method, which consisted of four phases.

The first screening of which scientific papers to consider out of the search results on Google Scholar or references of already found papers was based on the titles of the links. That is, links with titles that seemed promising were opened. The abstract of a paper was read to decide if the content of the paper should be reviewed. If the abstract gave the impression that the content of the paper could be relevant for this work, the paper was skimmed through. If the content was determined to indeed be relevant for this work, the paper was saved. For then deciding which of the saved papers to include in the report, the papers addressing similar topics were compared, and the papers of them that were evaluated as being the most relevant and of the highest quality were included. For some topics, there were not many papers to choose between. For those topics, all papers were included, regardless of the relevance or quality of the content.

Other resources than scientific papers that were included in this report mostly addressed either micro frontends, certain technical tools, or definitions of technical concepts. The ones about micro frontends were chosen based on if the author seemed to have a sound background in the software field and if the content of the resource was evaluated as being of high quality. For information about technical tools, mainly official web sites of the tools were chosen. For definitions of technical concepts, established standards were chosen as often as possible.

Some consideration was also made with regards to the year of publication. Resources with content related to technical theory such as properties of a specific framework were only considered if they were published in the year of 2015 or later. Other resources, for example those presenting established methods still used today, were considered even though they were significantly older than from 2015.

4.3 Implementation

Two prototype applications of the frontend of the web application described in *2.2 Existing Solution* were developed, one implementing a traditional SPA pattern and one implementing a micro frontends pattern. React was chosen to use as development framework because of the its popularity, as described in *3.8 React*, which meant that there would exist a lot of resources to get help from when problems were encountered. To implement certain components in the applications, the React library Material-UI [63] was used. The UIs were developed to be as similar as possible to the layout of the existing solution described in *2.2 Existing Solution*.

To be able to compare the two architectural styles of the implementations, they had to differ only with regards to their architecture. To ensure that this, the applications were implemented in parallel. A component was first implemented in the SPA. When it was finished, the code

was copied to the MFA and adjusted as little as possible to work in that solution. Below, development details unique for each prototype are described.

4.3.1 SPA Prototype

The base of the SPA prototype was generated by executing the Command-Line Interface (CLI) command shown in Listing 4.1 below, in which `npx` [64] is a package runner tool, `create-react-app` [65] is a tool for generating an SPA in React, and `spa-application` is what the generated application was named.

```
1 $ npx create-react-app spa-application
```

Listing 4.1: CLI command used to generate the base of an SPA called spa-application.

4.3.2 MFA Prototype

The MFA prototype was in addition to React developed using single-spa (see *3.9 single-spa*). An abstract architectural diagram was drawn to illustrate how the application would be split into micro applications. Which feature that would belong to which micro application was decided based on how tightly related features were in the UI. Architecture for a hypothetical backend was also included in the diagram. The diagram was adjusted, and details were added during the implementation as new discoveries were made.

The code base of the root application was generated by running the CLI command shown in Listing 4.2, in which `create-single-spa` [66] is a CLI tool for generating MFAs. The flag `--moduleType` controls which type of application is generated, which in Listing 4.2 is set to `root-config`, meaning that an application that will construct an application using other micro application is generated.

```
1 $ create-single-spa --moduleType root-config
```

Listing 4.2: CLI command used to generate the root application of an MFA.

The code bases of the micro applications were generated by running the CLI command shown in Listing 4.3, in which the `--moduleType` flag is set to `app-parcel`, meaning a micro application that will be included by a root application is generated. The `--framework` flag controls which frontend framework the micro application will use, which in Listing 4.3 is set to be React.

```
1 $ create-single-spa --moduleType app-parcel --framework react
```

Listing 4.3: CLI command used to generate a micro application to be included by the root application of an MFA.

4.3.3 Testing of Prototypes

To further ensure that the two prototypes did not differ any more than necessary, exploratory testing of the UIs of both prototypes were performed in parallel by the developer of the prototypes. First an interaction with one of the applications was performed, and then the same interaction was performed on the other application to ensure that the applications reacted in the same way. If something differed, the code bases were inspected, compared, and adjusted to react in the same manner. Additionally, the appearances of UIs were compared, and similarly, the code bases were adjusted if any inconsistencies were discovered.

4.3.4 Analysis Program

To extract code metrics from the implementations to be used in the mathematical model-based assessment, an analysis program was developed. The program was written in Python, and Python libraries glob [67], os [68], re [69], and NumPy [70] were used in the implementation. Only JavaScript files were processed. That is, files such configuration or style files were excluded from the analysis.

4.3.5 Testing of Analysis Program

To verify that the analysis program produced correct results, a test program running unit tests was developed using Python libraries unittest [71] and parameterized [72]. Files from the prototypes were chosen as test cases. Specifically, files that differed from each other in their content were chosen so that the test data would be diverse and cover a variety of implementation content.

4.4 Semi-structured Interviews

As a part of the attempt to collect data for answering the research questions, experience-based assessment was chosen as one method to use because of the arguments presented in *3.4.3 Experience-based Reasoning*. To make an experience-based assessment, semi-structured interviews were held with seven employees at the company described in *2.1 Origin of Work*. The interviewees were chosen based on the fact that they had experience of working with micro services in backend development, experience of frontend development, experience in software architecture, and/or experience of working as lead developers.

4.4.1 Data Collection

To create the structure and content of the interviews, other research papers about software architecture that included semi-structure interviews in their methods were studied as inspiration, and interview guidelines described in Seamans work (see *3.5.1 General Interview Guidelines*) were considered. An interview guide was created to be used during the interviews. The interviews were held in Swedish, but everything from the interviews that will be included in this report has been translated to English. Below, an overview of what each interview contained is provided. The full interview guide in English can be viewed in Appendix A.

Each interview took between about one hour and one and a half hour to perform and were done in distance-mode via Zoom. They were recorded using Zoom's recording tool so that they could be transcribed afterwards. At the beginning of an interview, the interviewee was informed about the interview structure, and that they were welcome to provide extensive answers, even if some of them were not direct answers to the asked question. After that, a brief overview of the background of the research was provided to the interviewee, together with a general description of what the purpose of the research was. If the interviewee did not have any previous knowledge of micro frontends, alternatively only brief knowledge, a walk-through of the concepts of it was provided.

When the background of the work had been presented, questions regarding the interviewee were asked. The following demographic data on the interviewee were collected:

- Current work role
- Years of work experience within the software field
- Possible knowledge about micro frontends

- Possible experience of working with micro services

Six out of the seven interviewees had experience of working with micro services in backend development. All interviewees except one had heard about micro frontends, but none of them had experience of working with it in the way it is described in *3.2 Micro Frontends*. The working roles of the seven interviewees can be found in Table 4.1. The number of years of work experience within the software field of the interviewees can be viewed in Figure 5.6, and the average was about 6.15 years.

Work role	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**
Backend developer	x	x	x	x	x	x	
Frontend developer		x		x	x		x
Architect		x			x		
Lead developer	x	x		x	x		x

Table 4.1: Work roles of the interviewees, where interviewees are represented with P1 – P7.

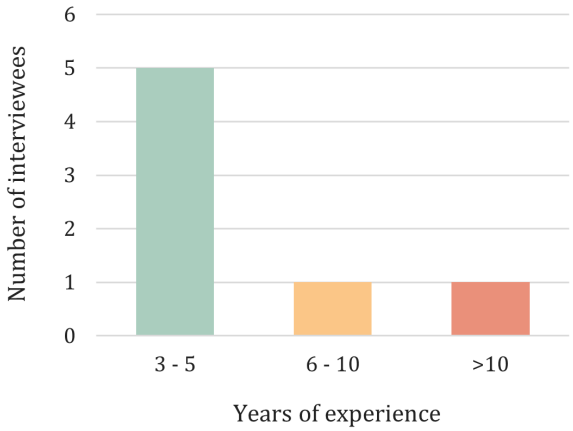

Figure 4.2: Years of experience within the software field of the interviewees.

When the demographic data had been collected, interviewees with experience of working with micro services were asked questions on their opinions on the concept. Amongst other things, the questions addressed what the interviewee thought were benefits and challenges with working with micro services.

The last series of questions that were asked addressed micro frontends. The first question asked about micro frontends was and open-ended question about the interviewees general, spontaneous thoughts on the concept:

- What are your spontaneous thoughts on micro frontends?

The purpose of beginning the micro frontends questions with asking a general, open question was to collect opinions of the interviewee without influencing their answer with how the question was asked. After the first question however, more specific questions regarding specific aspects of micro frontends were asked. Examples of such questions are:

- What are your thoughts on deploying frontend modules separately?
- What are your thoughts on the team structure ideas of micro frontends?
- What effects do you think there are on the testing of an application when it is implemented with the micro frontends concept?

Since the interviews were semi-structured, the interview guide acted as a base for the interviews but was not followed exactly. Some questions that were included in the interview guide were not asked explicitly since they were answered by the interviewee anyway. Sometimes when the interviewee brought up something that was considered to be of interest for the research when answering an asked question, questions that were not included in the interview guide were asked as follow-up questions. The order of when the questions were asked could also differ depending on what topics the interviewee brought up when providing answers.

4.4.2 Data Processing

Similar to in the interview studies described in *3.5.4 Data Processing*, the data processing phase consisted of four steps constructed with inspiration from the study. These are described below.

Step 1: Data Preparation

After the interviews, most of the content of the recordings was transcribed word for word. However, parts that were considered to be far outside the scope of the research objectives were excluded from the transcripts to save time. Some of the demographic data were recorded in a table, and some were recorded in a bar diagram. The average number years the interviewees had of working within the software field was calculated.

Step 2: Definition of initial topic categories

A number of initial topic groups were defined before the codification of the data began. In contrast to the second step in the study described in *3.5.4 Data Processing*, these topics were not considered to be main topic categories, but rather a base set of categories that would be expanded as the codification was performed.

Step 3: Data codification

The transcripts were read through with the purpose of tagging transcript parts with topics. The first step was to tag sentences or paragraphs as being about a specific topic. Multiple tags could be assigned to a transcript part. In addition to the initially defined topics, topics were also defined successively as the transcripts were read based on the contents of the transcript parts. After the transcripts had been read through the first time, they were read again. If transcript parts were discovered to fit topics that had been defined after they were read the first time, the topics were assigned to the transcript parts.

Step 4: Interpretation of results

After the data codification, informal summaries of all tagged transcript parts were created and grouped with other transcript parts addressing the same or similar topics. Lastly, all groups of informal summaries were compiled into one formal summary each, which were assigned a summarising topic title.

4.5 Mathematical Model-based Assessment

As a part of the attempt to collect data for answering the research questions, mathematical model-based assessment was chosen as one method to use because of the arguments presented in *3.4.4 Mathematical Modelling*. Out of the two mathematical models of modifiability that were found during the pre-study (see *3.7 Mathematical Models of Modifiability*), the SQMMA model was selected because the metrics used in the model presented by Dayanandan and Kalimuthu were judged to be too object-oriented specific to be used for React projects.

4.5.1 Data Collection

The data collection consisted of retrieving required code metrics for using the SQMMA model. Assumptions of the definitions of the metrics *AverageMethodSize*, *MethodInheritanceFactor*, *NumberOfChildren*, and *PolymorphismFactor* used in the model had to be made. These assumptions are described in *3.7.1 SQMMA*. As described in *3.8.1 React Components*, there are several reasons to use functional components rather than class components in React, which is why functional components were used when building the prototypes in this research. This however meant that the implementations were not class based. Therefore, to be able to derive metrics that involved class concepts in their definition, React functional components acted as equivalents to classes. States acted as equivalents to instance variables. The code metrics *MethodInheritanceFactor* and *PolymorphismFactor* were set to 0 since React does not have any equivalents to the object-oriented concepts used in those metrics.

As mentioned in *3.8.2 Composition*, composition is used rather than inheritance in React, and because of that, composition was used as equivalent to inheritance when calculating *DepthOfInheritance* and *NumberOfChildren* of the prototypes. These were retrieved manually since composition was used sparingly in the prototypes. Because this yielded values smaller than 0.001, the metrics were set to zero.

The rest of the code metrics needed to use the SQMMA model were retrieved by using the implemented analysis program. For the SPA, the program was run for the source code repository. For the MFA, the program was run for the source code repository of each micro application as well as the source code repository of the root application. The averages of the results derived for the different source code repositories of the MFA were calculated.

4.5.2 Data Processing

The code metrics derived during the data collection were used to calculate changeability and stability according to the SQMMA model using Equation 4.1 and Equation 4.2 (based on Equation 3.6 and Equation 3.7 in *3.7.1 SQMMA*) for both the prototypes. Then, the modifiability for both prototypes were calculated according to SQMMA using Equation 4.3 (earlier presented as Equation 3.8 in *3.7.1 SQMMA*). Lastly, the calculated modifiability values were normalised by dividing the values with the value for the MFA.

$$\begin{aligned} Changeability = &-0.13 * AverageMethodSize \\ &-0.16 * CouplingBetweenObjects \\ &-0.15 * 0 \\ &-0.13 * LackOfCohesion \\ &-0.14 * 0 \\ &-0.16 * NestedBlockDepth \\ &-0.13 * 0 \end{aligned} \tag{4.1}$$

$$\begin{aligned} Stability = &-0.20 * DepthOfInheritance \\ &-0.21 * NestedBlockDepth \\ &-0.19 * 0 \\ &-0.21 * ResponseForClass \\ &-0.18 * 1 \end{aligned} \tag{4.2}$$

$$Modifiability = 0.42 * Changeability + 0.58 * Stability \tag{4.3}$$

5 Results

This chapter presents the results of this study. They are presented chronologically according to how the method was described in the previous chapter.

5.1 Pre-study

A summary of the theoretical knowledge obtained during the pre-study are presented in *3 Theory*. The pre-study also resulted in decisions being made regarding which methods to use during the research work. It was decided that two prototypes would be implemented so that the architectural aspects of micro frontends could be compared with the SPA architecture. These prototypes would then be evaluated using a mathematical model of modifiability, and with an experience-based assessment method consisting of semi-structured interviews. These interviews would also aim to collect data valuable for answering research questions about the micro frontends concept in general.

5.2 Implementations

Below, the results from the implementation phase are presented. Descriptions of how the prototype implementations are structured and how they work are provided together with figures showing the architectural structures and UIs of them. The implemented analysis program is described as well as the unit tests used for ensuring the correctness of the program. No results, other than that it was ensured that the prototypes were as similar as possible, were derived from the manual testing of the prototypes, and therefore no results related to that part of the method will be presented.

5.2.1 SPA Prototype

The resulting UI of the implemented SPA prototype, which resembles the UI of the application described in *2.2 Existing Solution* as much as possible, can be viewed in Figure 5.1.

The SPA prototype is built as a traditional React SPA. The source code folder of the project contains two files called index.js respectively App.js, as well as two folders called pages respectively components. The app file contains the root component of the project, that is,

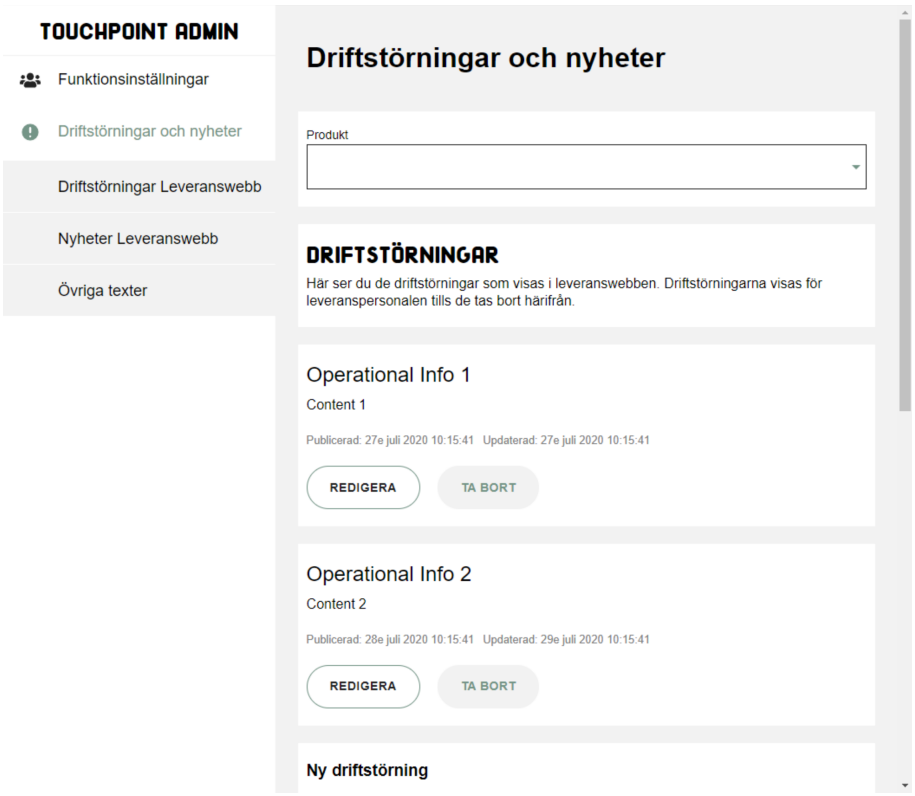

Figure 5.1: UI of the SPA prototype.

the component that will include all content of the web application. The index file renders the root component. The components folder contains all defined components used in the application, for example a button component or a text field component. The pages folder contains larger components that make up entire pages all related to a unique URL. The root component handles which page should be shown when visiting a specific URL. The page components use the smaller components from the components folder. The entire application is run locally on port 3000 by executing the CLI command shown in Listing 5.1, in which `npm` [73] is a package manager. An overview of the structure of the source code of the SPA prototype is shown in Figure 5.2.

```
1 $ npm start
```

Listing 5.1: CLI command to start the SPA prototype on port 3000.

5.2.2 MFA Prototype

The UI of the MFA prototype, which looks the same as the UI of the SPA, can be viewed in Figure 5.3. However, in Figure 5.3, the part within the green, dashed line marking is provided by the menu micro application, and the part within the orange line is provided by the info micro application.

The first version of the architectural diagram of the MFA, created before the actual implementation of the application, can be viewed in Figure 5.4. This was the diagram used as a basis for implementing the prototype, and as can be seen, it was decided to divide the application into three micro applications. One of them provided the menu of the application, and the other two provided the pages and components belonging to one of the two menu items in the menu as well as their belonging sub menu items. For example, the info application provided the

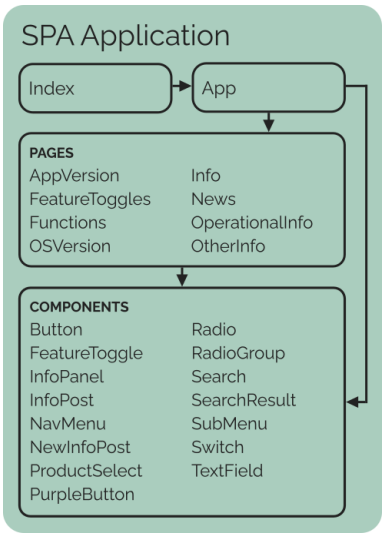

Figure 5.2: Overview of the structure of the SPA prototype source code.

pages and components related to the menu items *Driftstörningar och nyheter*, *Driftstörningar Leveranswebb*, *Nyheter Leveranswebb*, and *Övriga texter* shown in the menu in Figure 5.3.

The final version of the architecture diagram can be viewed in Figure 5.5. It includes more details about what components the root application and the micro applications contain, as well as dependencies that it was discovered were required for the entire application to function correctly. These dependencies will be described in more detail later. Note that everything not included in the presentation layer is hypothetical and not actually implemented. The MFA prototype ended up consisting of one root application and three micro applications. The root application contains three files called activity-functions.js, index.ejs, and micro-frontend-root-config.js, which are represented by *Index*, *Root Config*, and *Activity Functions* in Figure 5.5. The root config file registers and mounts the micro applications that should be included in the final web application. Which micro applications should be active at the moment is based on the current URL. The activity functions file contains functions for each micro application that returns true or false depending on the current URL. When the root config file registers the micro applications, it maps the functions in the activity functions file to the corresponding micro application. The index file contains HTML code that defines div elements for containing the micro applications, imports of shared dependencies such as react and react-dom, imports of the micro applications, and other relevant HMTL tags for defining for example which charset to use. Each source code folder of the micro applications contained an index file, a root component file, and a components folder. The applications called Settings Application and Info Applications in Figure 5.5 also had pages folders. The index files provide information to the root application about how to mount and unmount the micro applications. Amongst other things, it provides the root component which is defined in the root component file. Like in a traditional SPA, the root component includes components from the pages folder, which in their case include components from the components folder. Because the micro application for the menu, called *Menu Application* in Figure 5.5, does not contain a pages folder because all content of the menu is shown for the same URL, the component called NavMenu in Figure 5.5 is included directly in the root component. To run the application locally, the command shown in Listing 5.2 is executed for the root application and each one of the micro applications in their own project folder and with different arguments for the `--port` flag.

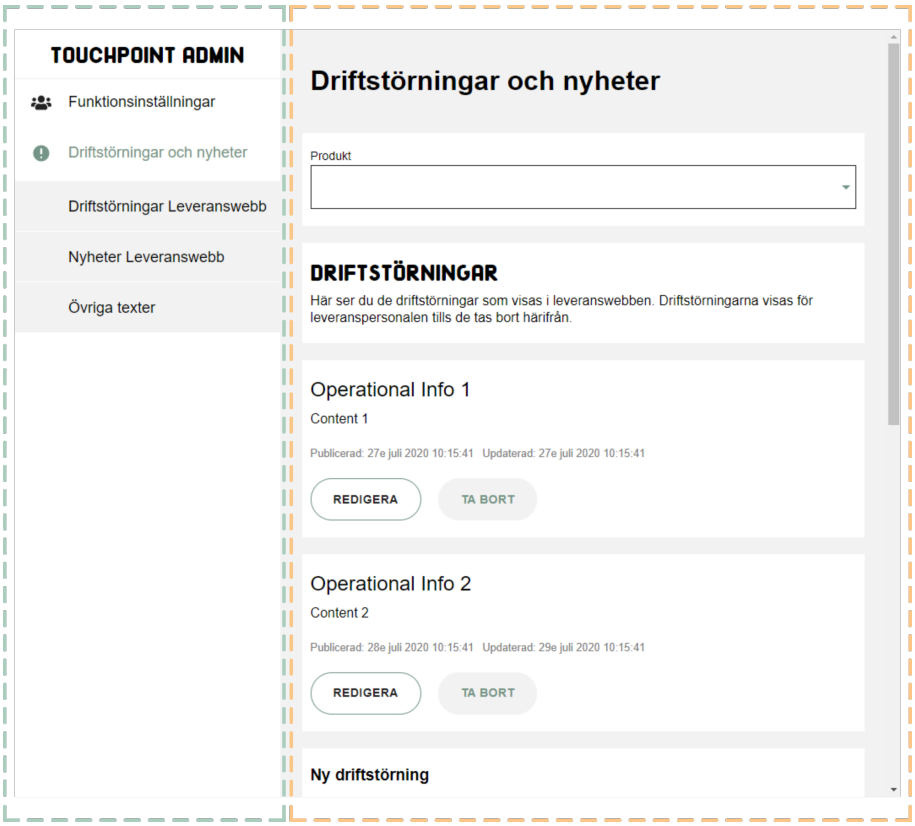

Figure 5.3: UI of the MFA prototype, where the part inside the green, dashed line marking is provided by one micro application, and the part inside the orange is provided by another.

```
$ npm start -- --port [PORT]
```

Listing 5.2: CLI command that can be used to start running either a root application or a micro application on the port specified by `[PORT]`.

As more was learned about the application, it was realised that both the settings part of the web application and the info part need access to certain data regarding products. For the MFA, this would mean that if a backend was provided to it, the settings application and info application would share one dependency. Apart from that, the settings application would rely on a backend microservice to provide certain data on companies owning the products, and the info application on a microservice to provide operational information about the products. These dependencies are indicated with arrows in Figure 5.5. Furthermore, there would have to exist dependencies within the data layer, since products are owned by companies, and the operational information is about certain applications. This is however not shown in Figure 5.5. The menu application would rely on the settings application and the info application to provide it with information on which links to include in the menu. This dependency is also indicated with arrows in Figure 5.5. Worth noting in the figure is that the settings and info applications have components with the same name. The looks and functionality of these components are identical.

One thing that had to be implemented differently in the MFA than in the SPA was the handling of changes in the UI in the menu application. In both the implementations, the menu item currently whose belonging page is currently open should have the font colour green, whereas the other menu items should have the font colour dark grey. This is shown in Figure 5.1 and Figure 5.3, where the menu item *Driftstörningar och nyheter* is in green. In the SPA, the root

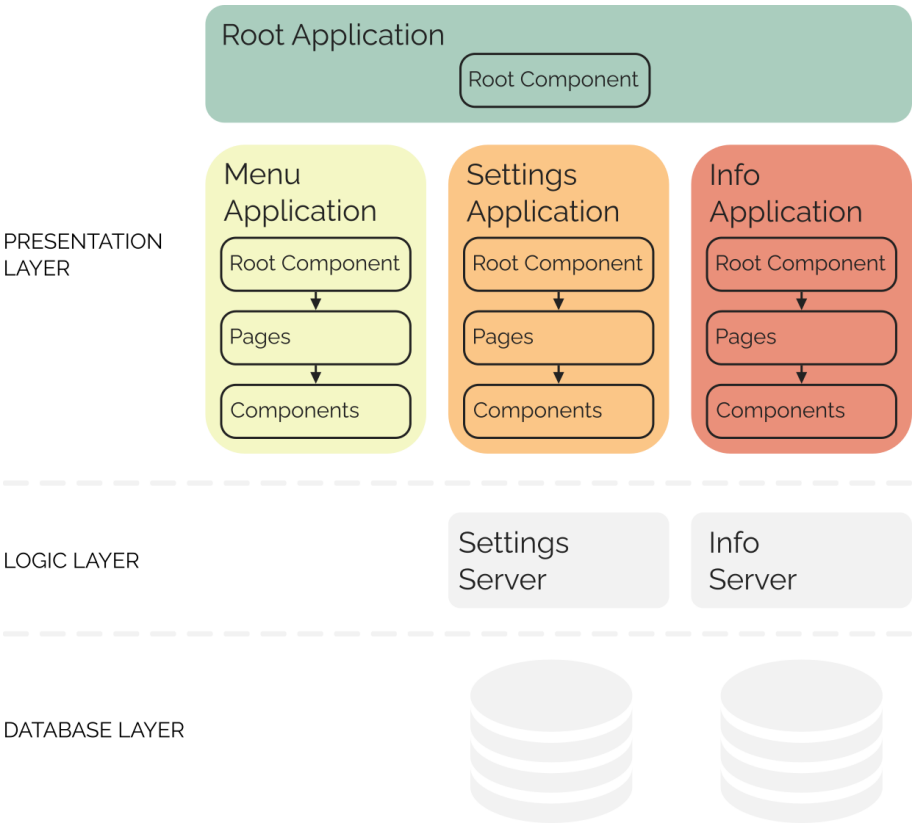

Figure 5.4: First version of the architectural diagram of the MFA prototype.

component passes a special prop called *location* to the *NavMenu* component that contains the current URL. The *NavMenu* passes this special prop to the *SubMenu* components. This prop is updated when a new URL is visited. Since a component is rerendered when a prop is updated, this means that the *NavMenu* component and the *SubMenu*s are rerendered when a new URL is visited, and in that way, the menu item corresponding to the new current URL is coloured green. In the MFA however, the *NavMenu* component is contained in one of the micro applications, the menu application, and therefore receives its props from that. Because the menu application is loaded from the root application, the special *location* prop in the menu application is not updated when a new URL is visited. This in turn means that the *NavMenu* and *SubMenu*s are not rerendered when a new URL is visited, and are not recoloured. To handle this, the *SubMenu* component contains a state that is updated when a menu item is clicked and a new URL is visited. Because a component is rerendered when a state is updated, this means that the menu items are recoloured to match which URL is currently visited.

5.2.3 Analysis Program

The analysis program consists of five functions for calculating required code metrics, which all uses help functions. An example of a function is `calculate_NBD`, which returns the *NestedBlockDepth* a file and can be viewed in Listing 5.3. All functions, including help functions, can be viewed in Appendix B.

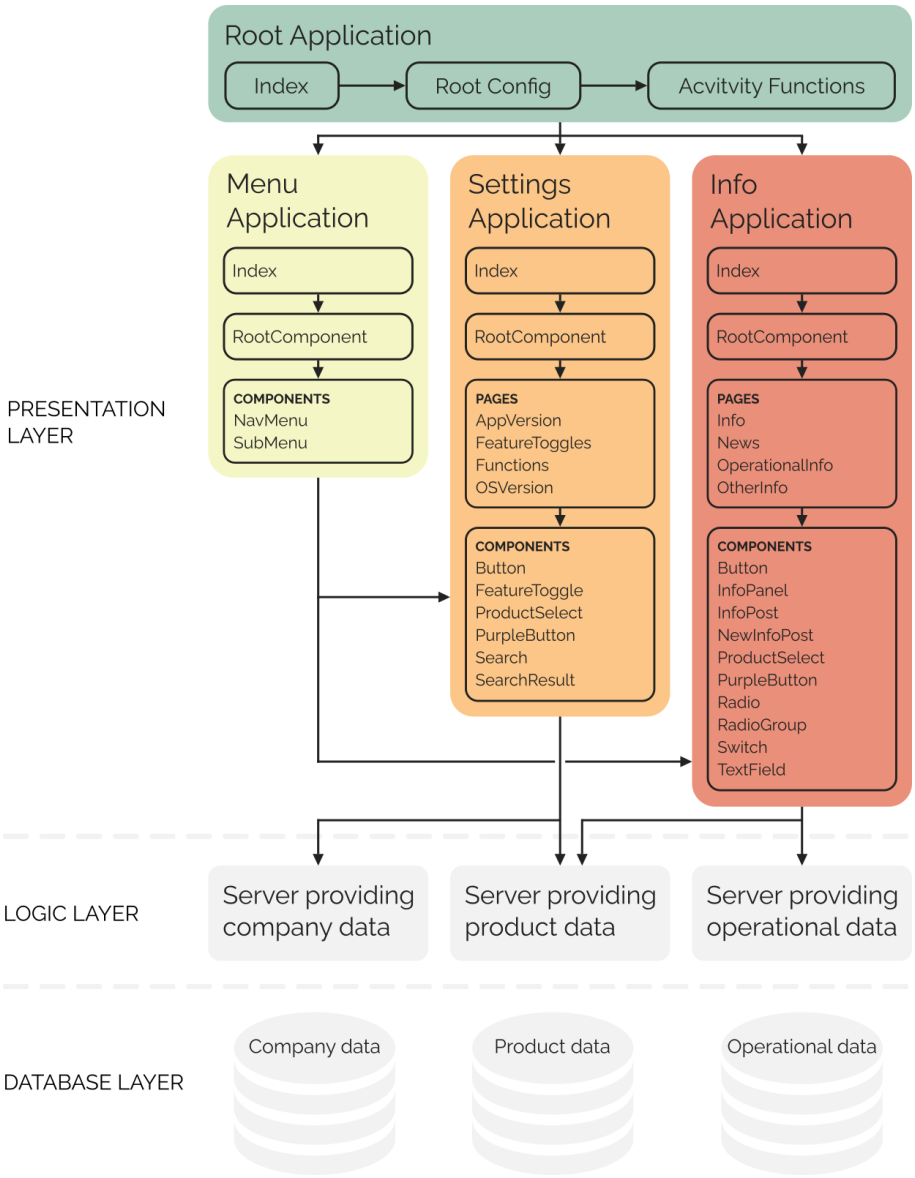

Figure 5.5: Final version of the architectural diagram of the MFA prototype.

```
def calculate_NBD(file_path):
    functions = get_functions(file_path,
                              include_component_function = False)

    total_NBD = 0
    for function in functions:
      max_depth = 0
      current_depth = 0

      function.pop(0)
      for line in function:
        current_depth = update_current_depth(line, current_depth)

        if (current_depth > max_depth):
          max_depth = current_depth

      total_NBD += max_depth

    return total_NBD / len(functions) if total_NBD != 0 else 0
```

Listing 5.3: Function `calculate_NBD`, which returns the *NestedBlockDepth* of a file.

5.2.4 Testing of Analysis Program

The test program for the analysis program consists of five unittest classes representing one test case each, which in total run 41 tests. That is, one unittest class, or test case, per function in the analysis program. One such test case can be viewed in Listing 5.4. The full test program can be viewed in Appendix C.

```
class AMS(unittest.TestCase):

  data_repository = '..\\test_data\\'

  @parameterized.expand([
    ['activity-functions', 1.333],
    ['App', 0],
    ['Button', 0],
    ['FeatureToggles', 5],
    ['index', 0],
    ['InfoPanel', 0],
    ['NavMenu', 7.666],
    ['NewInfoPost', 1],
    ['Search', 3]
  ])
  def test_AMS(self, name, expected):
    AMS = calculate_AMS(self.__class__.data_repository + name + '.
  js')
    self.assertAlmostEqual(AMS, expected, places = 2)
```

Listing 5.4: Test case that verifies the output of the function `calculate_AMS`.

5.3 Semi-structured Interviews

In this section, the results from the interviews are presented. First, a description of the results from the data collection is given. Then, the results from the data processing are presented.

5.3.1 Data Collection

The semi-structured interviews resulted in five hours and 46 minutes of recorded material. The contents of the recordings are demonstrated as summaries of some selected parts of the transcripts considered to be relevant for the research and are presented categorised by defined topic groups in *5.3.2 Data Processing*.

5.3.2 Data Processing

The transcription of the recorded interviews resulted in a total of 40 A4 pages of material containing about 25 000 words. The demographic data of the interviewees are found in *4.4.1 Data Collection*. Initially, before the coding of the data began, six topic categories were defined. These were the following:

- Microservices
- Micro Frontends Team organisation
- Micro Frontends Architecture
- Maintainability and Modifiability
- Effect on Testing
- Suitability in Certain Applications

After the coding of the data and the interpretation of the results however, seven adjusted topic groups had been defined. These were the following:

- Full Stack Teams Owning Features
- Standalone Applications
- Separate Techniques Side by Side
- Effect on Further Development
- Effect on Testing
- Suitability in Certain Applications
- Challenges With Isolated, Non-Dependent Teams and Micro Applications

The final topic groups are described below together with the data belonging to them.

Full Stack Teams Owning Features
This topic group contains interview answers related to the micro frontends concept that is that teams are split on features in the UI rather than on technique, and that they therefore work full stack.

> *"From my experience, when you work with microservices, it is easy to grasp to whom each responsibility belongs. Few things slip between the cracks."* – Interviewee

One interviewee said that the fact that a feature service is owned by one team probably results in that team knowing a lot about that feature because the scope is narrow, and they have knowledge of the service all the way from the presentation layer to the data layer. As when working with microservices in general, another interviewee described, the responsibilities of specific teams are very clear and therefore the risk of a task being forgotten about is smaller than when one team is working with one large application. In micro frontends, where a feature in the frontend is owned by one team, it is likely to be very clear which team is responsible for what.

A third interviewee thought that it was a good idea to work with an application from the top layer to the bottom. The reason was that it could be easier to have a common goal within all developers working on the application, compared to if one team works with the frontend part of the application, and another works with the backend part. The interviewee's experience was that it is common that there are misunderstandings between frontend and backend teams when it comes to what they need from each other. If all developers needed to change something in an application are members of the same team however, whether the team consists only of full stack developers or with both frontend and backend developers, communication might be facilitated, the interviewee speculated. A fourth interviewee also highlighted that having full stack teams in control of the entire stack flow probably means that the developers have more knowledge of what to expect from different modules of the stack than if the modules are developed by another team. The interviewee continued that there probably is a smaller risk that the team must sit idle and wait for a dependency to be finished by another team. If something is wrong with a module, they never have to disturb another team that would have to adjust their planning to the newly discovered problem. Instead, all problems can be handled internally. A fifth interviewee thought that developers in general would enjoy belonging to full stack teams since it could result in more varied work tasks compared to in frontend respectively backend teams.

> *"You cannot think of one team consisting of specific team members. If you do, the members will sit on their hands when the when they run out of work tasks."* – Interviewee

An opinion of one interviewee was that it would not work to have teams owning specific features if the teams were not flexible with regards to which members belonged to the team, because otherwise teams could run out of tasks. The experience of the interviewee was that it varies which features of an application are in need of work at the moment. In some time periods, a feature does not need any work at all, and in other, it needs a lot of work. Therefore, the interviewee only considered it possible to split teams with regards to feature if the teams were virtual. This would mean that team members could jump between feature teams depending on which features requires work at the moment, and that a team could be paused during the time when a feature did not need new development. Specifically, the interviewee thought this would make developing new features easier.

Standalone Applications

In this topic group, benefits of applications being isolated and deployed on their own described by the interviewees are included.

> *"You will be able to do small releases of an application often, only for the parts that are involved in the change. If something breaks in one part of the application, you can fix it and redeploy it quickly since you do not have to redeploy the entire application. And that is pretty nice."* – Interviewee

Several interviewees argued that as when using microservices in a backend, having a web application, including the frontend part, split up into several micro applications that are isolated and deployed separately means that one micro application can break without affecting the others. If one micro application is overstressed, the others are not. In the frontend, two of them describe, this could mean that a feature that breaks can disappear from the UI, but the other features continue to be available to the user. Instead of having downtime on the entire application, there is only downtime on one feature. However, one of them continued, this relies on that the micro applications actually are isolated and that there are no, or very few, dependencies between them. Another interviewee highlights the benefit of enabling small releases of certain parts of a web application. Developers will not need to redeploy an entire web application just because a small part of it has been updated.

Separate Techniques Side by Side

In this topic group, opinions of the interviewees on micro frontends allowing separate techniques to be used side by side in the frontend of an application are presented.

Two interviewees argued that some programming languages or frameworks are better suited for certain purposes, for example with regards to performance. They therefore had a positive view of that the micro frontends concept allows for usage of different techniques in different micro applications since that means that the choice of technique can be optimised for a certain purpose. Another strength with the choice of technique being flexible is that the choice can be adjusted to the competence within the team, two interviewees argued. One of them continued that if a company has few developers available with different framework skills, these can be composed into teams that use different techniques but that still work on the same application.

One interviewee was of the opinion that being able to implement micro applications using different techniques was a key feature of the micro frontends concept. At the same time, the interviewee expressed that it could be challenging if a team member should be able to jump to another team when there are no current tasks for their service. The other service could be implemented using other techniques, design patterns, or similar, than the original one, and then it would probably take some time for the member to gain enough insight into the other service to be able to start being productive. Two other interviewees also brought this up as a possible problem, and one of them thought that it in general would be better to use the same techniques for all features, at least in the frontend. A fourth interviewee also addressed the

subject of having to deal with new techniques when jumping between teams but did not see it as a big problem if the developer had a genuine interest for programming, since learning new things is part of a programmers job, and most people enjoy variation.

> *"Let us say you visit a web page and it takes ten seconds to load, because your browser has to load React, and Angular version 1.2, and Angular version 8 and version 9, and they are supposed to work side by side. That is a severe drawback."* – Interviewee

Another concern was brought up by an interviewee that regarded a possible problem with performance. If an application using the micro frontends pattern uses multiple frontend frameworks, such as for example React, AngularJS, Angular v.9, and so on, it could affect the performance of the web application remarkably since all frameworks have to be loaded into the web browser, the interviewee described.

Effect on Further Development

In this topic group, opinions of the interviewees when informed that the micro frontends concept is supposed to have a positive effect on further developments are included.

When asked about their opinion regarding the claim that adopting the micro frontends concept facilitates further development on a web application, three interviewees explained that as with all microservice architectures, micro frontends architecture allows incremental upgrades of an application. That is, they explain, if there is a need to change the implementation tools used in a web application to more modern tools, or to change from an older to a newer version of a tool, this can be done for one micro application at a time. The newer versions of the micro applications can then run side by side with the micro applications that are still using the old tools, until they have all been upgraded. This would make it easier to upgrade an MFA than a monolith.

Another interviewee responded that yes, further development could be facilitated by the fact that the micro frontends concept allows for small teams, which in general is positive. If the micro applications had no dependencies between each other, the interviewee continued, development is facilitated since a team can work isolated with their micro application and know that if it works in their environment, it will work in production, regardless of whether the other micro applications work or not. This was also the thoughts of another interviewee, who also described that this is the case for all microservice architectures. Since all micro applications are running isolated, and should not have any dependencies between them, making changes in one of them does not affect the others, which facilitates for the developers. Further, the interviewee explained that in a monolith, a part of a codebase that was developed a long time ago and that is working well can break just because someone is making changes in a new part of the codebase. This is not the case for microservice architecture, in which codebases are isolated. A third interviewee also expressed that it is usually smoother to add features by adding a new service, and a fourth that having smaller codebases, which could be a result from using the micro frontends architecture, makes it easier to work in each small codebase. A fifth expressed a similar thought, and additionally said that working with smaller version control repositories that do not get pushed to by a large number of people simplifies the handling of the repositories. For example, it will probably result in fewer and smaller merge conflicts.

> *"In the web community, everything moves faster. I think things move fast in the Java community, but they move a lot faster in the web community. So imagine a problem is discovered with a feature that was developed three years ago by someone who is moved to another company, and is implemented in a toolkit you have never seen before. It is the same for all micro architectures. If you let something lay and rot for too long, it will become a problem."* – Interviewee

On the other hand, one of the interviewees explained, things move fast in the web community, and what is considered standard tools can change fast. If a micro application in an MFA is working well and need little or no maintenance and new development, it can pretty much be untouched during a couple of years. Then, when there finally is a need for a change, the competence required for modifying it may have disappeared from the company, which can make it harder to implement changes. It might be developed with a tool that is not any longer considered standard and therefore is not known by the current developers, and no one has an insight in the code because no one has worked with it. This can differ from if a feature in a monolithic implementation is not modified for a long time, because then there is a greater chance that the feature is similar to the rest of the monolith, and developers might at least have had to consider the code before implementing something new in the same code base. The interviewee continued explaining that this can be a problem for all microservice architectures, but that there may be an even greater risk within web development because of the rapid development of web techniques. This concern was also raised by another interviewee.

Another interviewee expressed that setting up the development environment was what the interviewee thought generally was the most challenging part of being a developer. Therefore, if developers should be able to jump between teams and work on other micro applications, it was important that there were general agreements between teams on how to configure setup tools so that how to use them would not differ very much. Otherwise, setting up a local environment might take an unnecessary amount of time if a team member has joined a new team, and this would contradict possible facilitative properties of micro frontends.

> *"I cannot say either yay or nay [regarding if further development is facilitated by using micro frontends]. I am not sure it has that effect. It feels like a lot of things affect that."*
> – Interviewee

One interviewee emphasised that a lot of factors play a role when it comes to how easy or hard it is to make changes in a web application. If you would compare a web application implemented in a messy manner with a web application that uses micro frontends where everything is very modular and isolated, then it is probably easier to modify the MFA. However, the interviewee continued, if you compare the MFA with a well written SPA that is implemented in a modular manner, then which one is easiest to modify is not that obvious to determine.

Effect on Testing

In this topic group, interviewees' thoughts on how testing is affected by adopting micro frontends are described.

The experience of one interviewee was that the more independent and well-defined components a system has, the easier it is to handle the components, and therefore to test them. The rule of microservice architecture that is to have as isolated services as possible with as distinct responsibilities as possible indicates that using the architecture can facilitate testing, the interviewee argued. Another interviewee also explained that when working with microservices in backend development, the interviewee had experienced that testing many small services was easier than testing one large monolith.

A third interviewee however had, when working with microservices in backend development, experienced that there often ends up existing many dependencies between services which results in a need for many integration tests. The interviewee thought this makes testing of a microservice architecture more troublesome than if all code exists in a singular codebase. This opinion was shared with two other interviewees, who explained that communication between microservices results in rather complex testing.

> *"Of course the testing will be affected, you will have to test the parts of the application that involves including the micro applications. You will definitely have to challenge that implementation. Will a micro application actually be included where it should be?"*
>
> – Interviewee

Two interviewees also insisted that the final, put together version of an MFA including all micro applications needs to be challenged and tested. Certain functionality of the web application needs to be verified to be working correctly when all micro applications are running side by side, for example that the correct micro application is included in the correct place at the correct time, or that the UI looks good when all micro applications are included. One of them specifically addressed the case of having an MFA to which the user can log in. In that case, there must be tests verifying that all micro applications have access to the user credentials at the correct time, and that indicates that the tests must be run on the final, composed MFA.

Suitability in Certain Applications

Ideas of the interviewees on when the micro frontends concept is profitable to use, and when it is not, are included in this topic group.

Four interviewees talked about that application size is a highly relevant factor when evaluating the suitability of implementing the micro frontends patterns in specific applications. They all described that the pattern can be valuable if an application is so big and contains so many different parts that it is hard to handle, maintain, and get a good overview of by one single team. If a split up of the application would result in very few micro applications, two of the interviewees explained that they thought that it would be unnecessary to distribute these micro applications between different teams, at least if it was possible for one team to handle the entire application. They related this to the fact that the micro frontends architecture in a way results in overhead in the form of deployment configuration. Another one of the interviewees reasoned in a similar manner by describing that for a simple application the micro frontends concept would result in unnecessary complexity when it comes to deployment. Yet another interviewee mentioned that the overhead in microservice architecture is worth to acknowledge when considering using the architecture. However, one interviewee pointed out, it is highly relevant to discuss whether the application will grow bigger and include more features in the future. Further, the interviewee explains that this is something that often can be answered in the beginning of the project. In that case the micro frontends pattern might be a suitable architecture pattern for a future version of the application, and then it might be worth the implied overhead to implement the application as micro frontends from the start. That will make it easier to add features side by side to the existing micro applications in the future, since the base of the architecture is already implemented.

> *"You have to ask yourself if it is worth it to implement an application as an MFA if it is supposed to have only two features. At the same time, no it may not be worth it, but you have to ask yourself, is there is a risk that we will evolve the application to a bigger one in the future?"*
>
> – Interviewee

Three interviewees all mentioned that they did not think that the micro frontends concept would be suitable if the features that an application would be split up into perform very little work and have few responsibilities. They felt that it would be unnecessary to have one team maintaining and developing such a simple feature service, and that the micro frontends concept would only increase the complexity of an otherwise relatively simple application. One of the interviewees specifically thought that having a micro application running a menu in a web application would be superfluous. Instead, the three interviewees thought, the concept is suitable to use if the features are complex. At the same time, another interviewee stated, the microservices should not be so complex that it is hard to summarise what their

responsibilities are. An example of a suitable application for micro frontends provided by one of the interviewees was portal solutions such as Microsoft Azure or Google Cloud Platform, where the applications consist of a portal that brings together many isolated, unrelated services for the user. The services can be rather complex and provide a lot of features for the user. The portal only controls user credentials and redirecting to different services, and do not know much about the services themselves. The services may even be provided by other companies than the owners of the portals, in these cases Microsoft and Google.

> *"When there appear too many 'if:s', it becomes complicated. ... Then you get a lot of dependencies between the services, and then I do not see the point of building them as microservices. Because they are not microservices, in that case they are a macroservice, a distributed monolith, one could say."* – Interviewee

One interviewee thought that micro frontends are fitting to use when a web application should do many different things and have many responsibilities, or even if it should contain many different services, similar to what is provided by for example Microsoft Azure as mentioned earlier. Another interviewee thought the micro applications could even be provided by different companies. An example this interviewee brought up was web applications providing mini games, where the games are unrelated to each other. The interviewee argued that the concept of micro frontends in general is a good tool for a company to use when outsourcing work to several different consultancy firms. Different companies work in different offices, may not use the same techniques, and do not have the same background in general, which can mean that it is challenging for them to work together or force them to work with the same techniques and work methodology. By using micro frontends, the interviewee continued, the different consultancy firms do not need to share infrastructure or cooperate regarding the same codebase via version control tools. This benefit with micro frontends does however rely solely on whether the services are completely isolated. If it is not possible to divide the web application into isolated micro applications that are not too dependent on each other, it is not worth using the micro frontends concept, was the opinion of one interviewee. If there are too many dependencies between the micro applications, the benefits, such as one micro application being able to break without affecting the others, of the concept are lost. In that case, time have been wasted on setting up the rather complex architecture of a microservice architecture, the interviewee finished.

One interviewee argued that micro frontends probably should not be used if there is a tight deadline or a small budget because of the implied overhead.

Challenges With Isolated, Non-Dependent Teams and Micro Applications

In this topic group, potential challenges with having isolated, non-dependent teams and micro applications expressed by the interviewees are presented.

As already mentioned several times, many interviewees emphasised that to be able to achieve the benefits of micro frontends, the micro applications have to be isolated and independent of each other, at least to a far extent. Several of the interviewees with experience of microservices in backend development said that how communication between services should be done is often not trivial to solve, and that it is challenging to grasp which data are needed in which microservice. One interviewee described that this often is solved by letting each service have an API to which other services send requests, but further explained that this can result in the creation of a lot of network connections and three-way handshakes if there are many dependencies between the services. This in turn can of course affect the performance of the application and extinguishes the benefit of microservice architecture that is that if one service breaks, all other services are still okay. A risk when making the design choice to use a microservice architecture that another interviewee addressed is that it later turns out that there are too many dependencies between the defined microservices, which means that they should

not be microservices at all. In that case, the architecture is unnecessarily complex because of extensive deployment configuration that would not be needed in a monolith solution. Also, it could be harder to find the cause of a bug compared to if you only have one large code base, since the bug could be located in any microservice code base because of the many dependencies.

For the micro frontends concept to work, one interviewee argued, there needs to be a well-defined mapping between the frontend and backend. The best case would be to have a one-to-one mapping. If a frontend module starts having to many mappings to different modules in the backend, then the architecture probably needs rethinking. Another interviewee thought that having for example a menu as a separate micro application would not work very well, since it is dependent on the content of the rest of the web application.

However, several interviewees brought up challenges regarding the features in micro frontends being completely isolated and non-dependent, both when it comes to the teams owning and developing the features, and the micro applications providing the features.

One issue raised by an interviewee was the case that a user should be able to log in to a web application implemented with the micro frontends architecture. Generally, when a user logs into a web application, the user credentials are used in several parts of it. In that case, the interviewee continued, the micro applications need to have a mutual view of who the user is, which means that they need to trust a provider of a token. This indicates that the services communicate and collaborate in some way.

> *"If you for example have two microservices that will do one request each to a database to retrieve data, and compare that to if you have one monolith that retrieves the data only once, well, then there will be a larger overhead when it comes to for example transactions against the data layer."* – Interviewee

Two interviewees brought up possible issues regarding data used in an MFA. Often, the same data source is used in several places in a web application, and if the application consists of several microservices, it is probable that the same data source is needed in multiple services. Requesting data can be expensive in terms of performance as well as expenses, and therefore, performing multiple requests to retrieve data from the same data source is not desirable, the interviewees explained. In a monolithic solution, one of them continued, where one team owns the entire solution, the team will make sure that only one request is made to provide data to multiple modules of the application. In a micro frontends solution, where teams work isolated from each other and with different parts of the application, this could be a problem. The teams need to communicate about this, the interviewee states.

Another challenge with working in isolated teams with isolated applications is the handling of non-functional requirements which becomes very important in the context of micro frontends, one interviewee argues. The interviewee mentions requirements on browser support as an example and explains that it becomes extra important to be clear about such requirements when the frontend is split up on different teams. Otherwise, the result could be a web application consisting of micro applications with support for different browsers.

Having different teams working in parallel with different micro applications could result in a variation in code quality across the application, two interviewees speculated. In a monolith consisting of only one code base, it might be easier to control that the quality of the code is of high standard from the beginning, compared to if you have very many code bases managed by one team each. This could make it extra important to define clear requirements on the quality of the code in the beginning of the project, one interviewee hypothesised.

> *"The most important thing of all is customer experience, user experience. How does it affect the user experience if you have containers in the UI with different looks, different styles, different types of error messages, different frameworks in the frontend?"*
>
> – Interviewee

One interviewee thought that the concept of micro frontends raises questions on how to cooperate between teams when it comes to aspects of an application that cannot be decided by the individual team. One such aspect that was mentioned was UI design, such as choice of colours and font types, which should be consistent throughout the application. Another was structure of error messages to the user, which should also be consistent. A third was how to handle popups from micro applications that should be able to cover other micro applications as well. The interviewee thought that there should be a dedicated user experience team that communicates with all teams so that the division of teams does not affect the user experience negatively.

Three interviewees were sceptical to the idea that teams could be completely isolated from each other when working on the same web application. At least it would be hard if the micro applications have to communicate with each other, which the interviewees described often is the case. One of them explained that even though the teams can write documentation on how other services should communicate with their service, there are often errors in the documentation, and they can easily become outdated. In those cases, teams need to be prepared to communicate with each other. The interviewee also expressed that a team probably wants to know if another team makes significant changes to a web application that they are both working on, even if the changes are made in parts not owned by the first team. A fourth interviewee on the contrary thought that teams working side by side with microservices in backend generally are able to work pretty isolated from each other. Even if some minor communication is needed because of required communication between the services, this was not a major problem, the interviewee stated. A fifth interviewee said that even though the need for communication between teams is small, it may be a shame to lose some communication in the form of knowledge sharing between the teams working with the same web application. Also, it may facilitate for developers to jump between teams if they have a minor clue about what other teams are working on.

5.4 Mathematical Model-based Assessment

The results from mathematically modelling the prototypes using the SQMMA model are presented in this section. First, the extracted, required code metrics values are provided. Second, the calculated values for the changeability, stability, and modifiability of the prototypes are presented.

5.4.1 Data Collection

The output from the analysis program included in Appendix B are presented in Table 5.1 for the SPA implementation and Table 5.2 for the micro frontends implementation. The values are rounded off to three decimals.

5.4.2 Data Processing

The results from calculating the SQMMA's definitions for changeability, stability, and modifiability of the SPA implementation are presented in Table 5.3, and of the micro frontends implementation in Table 5.4. The normalised modifiability score for the MFA was 1, and for the SPA 1.314. All values are rounded off to three decimals.

Metric	Value
Average Method Size	4.074
Coupling Between Objects	3.480
Depth of Inheritance Tree	0
Lack Of Cohesion	0.105
Nested Block Depth	0.113
Number Of Children	0
Response For Class	1.400

Table 5.1: Code metrics extracted from the SPA prototype by running the analysis program.

Metric	Value
Average Method Size	3.185
Coupling Between Objects	1.422
Depth of Inheritance Tree	0
Lack Of Cohesion	0.023
Nested Block Depth	0.135
Number Of Children	0
Response For Class	1.568

Table 5.2: Code metrics extracted from the MFA prototype by running the analysis program.

Metric	Value
Changeability	-0.561
Stability	-0.474
Modifiability	-0.510

Table 5.3: Calculated values for changeability, stability, and modifiability of the SPA prototype when using the extracted code metrics in Table 5.1 according to the SQMMA model.

Metric	Value
Changeability	-0.222
Stability	-0.509
Modifiability	-0.388

Table 5.4: Calculated values for changeability, stability, and modifiability of the MFA prototype when using the extracted code metrics in Table 5.2 according to the SQMMA model.

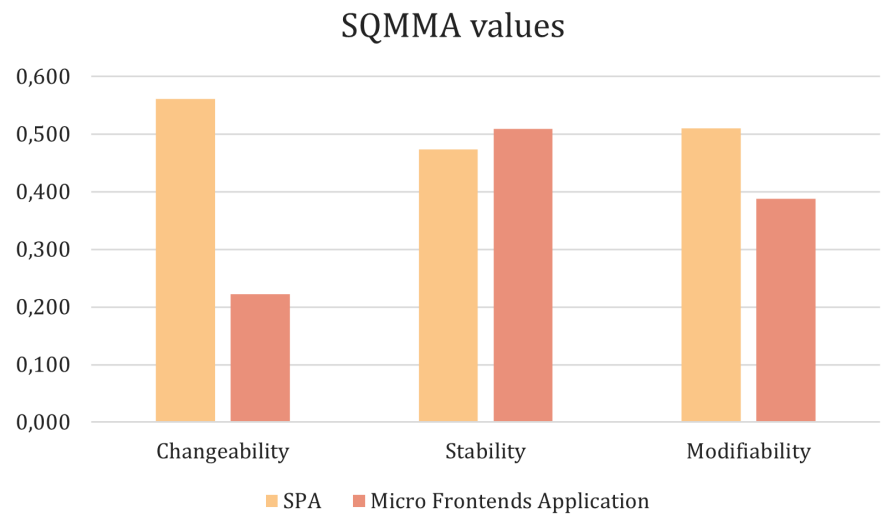

Figure 5.6: Absolute values of the calculated changeability, stability, an modifiability of the SPA prototype respectively the MFA prototype.

6 Discussion

This chapter contains discussions on the results, the method, and the wider context of this study. First, selected parts of the results are commented and compared with findings in the literature from the pre-study. Then, the method is critically discussed with special regards to replicability, reliability, and validity. Lastly, a discussion of how this work is relevant in a wider context.

6.1 Results

In this section, the results of this study are discussed, and some of them are also related and compared to statements made in the literature described in *3.2 Micro Frontends*. First, some interesting aspects of the implemented prototypes are explained and reasoned about. Then, some of the findings from the conducted interviews are discussed. Specifically, thoughts and opinions from the interviews not explicitly providing answers to some of the research questions, but rather implying things that could be used to answer the questions, are debated.

6.1.1 Implementations

As described in *5.2.2 MFA Prototype*, some components implemented in the settings and info applications are identical, and therefore, there exists code duplication in these micro applications. The alternative would have been to implement some sort of library containing the shared components which both the settings application and info application would have access to. This would resolve the problem of duplicate code but would instead mean that the micro applications would have yet another shared dependency. It could be argued that because only three components are identical, and the sizes of these components are relatively small with an average of about 35 lines of code, the profit of decreasing the code duplicity between the micro applications would be so small that it would be unnecessary to introduce a shared dependency. Further, implementing an external library would also require performing requests to that library, which affects the performance of the application negatively.

Because the division of the micro applications was not based on whether features would use the same data, the info application and settings application have to share one data dependency. This contradicts the idea described by Geers [8] that each micro application should provide a full stack solution from the presentation layer to the data layer since the

shared data dependency cannot be isolated to belong to only one micro application. One can ask whether it would be possible to avoid this shared data dependency by creating the division of the micro applications based on what data they use instead of on how tightly related they are in the UI. This however raises a question on how to handle the case when two features in the UI share the same data dependency but are supposed to be placed at different positions in the UI. With the single-spa solution used in this work, micro applications are placed in specific areas in the UI to which they are restricted. In the case of distributing one micro application over more than one area, a micro application would have to be able to provide several components to the root application, and the root application would have to know where to place each component. Currently, no such solution is described as possible in the single-spa documentation [61].

6.1.2 Semi-structured Interviews

A lot of thoughts and opinions of the interviewees can be related to facilitation of performing changes in software in micro frontends projects. As presented in *5.3 Semi-structured Interviews* in the results, one interviewee pointed out that using micro frontends could imply that teams become experts on their micro application because of the narrow scope and control from presentation to data layer. Another thought was that having all developers working on the same stack in a single team could help avoiding misunderstandings and facilitate communication between frontend and backend developers when implementing new features in an application, since communication about what the frontend needs from the backend and vice versa can be done within the team. Exactly this benefit has also been described by Geers [8]. A third interviewee also highlighted that having full stack teams working from top to bottom with an application meant that the developers probably have good knowledge of what to expect from every module in the stack, compared to if the modules were developed by other teams. The interviewee continued that there is a smaller risk that a developer would sit idle and wait for a dependency to be finished if the developer responsible for that dependency is within the same team than if from another team. Developing new features in an application could very well be facilitated by the strengths expressed by these three interviewees. If the developers of a team possess extensive knowledge about how their application works from top to bottom and communication needed for modifying the code can be done fast within the team, it is reasonable to assume that new development can be done faster than if the teams were split on frontend respectively backend teams. This supports Geers's [8] description of micro frontends facilitating performing changes and further development in an application. Another thing that could contribute to micro frontends facilitating further development in an application is the enabling of small releases of certain parts of an application, which one interviewee raised as a strength of micro frontends. Instead of having to redeploy an entire web application, the interviewee explained, only the new part needs redeploying, which reasonably makes it significantly easier.

Some opinions and thoughts on modifiability in micro frontends expressed by the interviewees agreed with the claims in the presented literature in *3.2 Micro Frontends*. Similar to what Geers [8], Jackson [6], and Myers [13] describe, interviewees explained that as with all microservice architectures, micro frontends allow for incremental upgrades of an application. It is legitimate to assume that upgrading one part of an application at a time is less risky than upgrading all parts at once and that it therefore can facilitate the upgrade. Two interviewees pointed out that having smaller codebases makes it easier to work in each independent codebase than working in one large, as was also described by Geers [8] and Jackson [6]. Additionally, Interviewees brought up possible benefits of micro frontends that are not brought up in the studied literature in presented in *3.2 Micro Frontends*, but that the interviewees thought could facilitate performing changes in software. As an example from the results, one interviewee described that working with smaller version control repositories that do not

get pushed to by a large number of developers simplifies the handling of the repositories, probably resulting in fewer and smaller merge conflicts.

Other benefits of micro frontends, not related specifically to modifiability, that were not brought up by the literature presented in *3.2 Micro Frontends*, but that the interviewees thought possible are also presented in the results. For example, one interviewee thought developers would enjoy belonging to full stack teams since it could result in more varied work tasks, and others thought it positive that the choice of frameworks to implement features in a frontend application with could be optimised based on the requirements on the feature.

When it comes to potential risks to acknowledge when starting a micro frontends project, many interviewees expressed ideas that are worth discussing. One idea of an interviewee that could be thought of as a possible risk that should be considered when adopting micro frontends could be the that of teams running out of tasks to work on for their assigned feature. The interviewee further described that one way to handle this risk was to consider the teams to be virtual in the sense that developers can jump between teams. This could however eliminate the strength of micro frontends described by Geers [8] that teams become experts on their features. Further, several interviewees thought that allowing team members to jump between teams could become a problem if different features were implemented using different techniques since it could mean that it would take some time for the new team member to get productive. This could also be considered a possible risk worth acknowledging when starting a micro frontends project.

Another idea that an interviewee had that is a risk worth considering if using different techniques in at least the frontend of an MFA is that of issues with performance. As the interviewee described, multiple frontend frameworks can be heavy for a web browser to handle, which can affect the performance significantly. Performance was also brought up by the two interviewees addressing the topic of micro applications using the same data. As described earlier, they explained that performing multiple requests for accessing the same data is very unnecessary and can affect the performance negatively. Therefore, a potential risk to acknowledge in a micro frontends project is that of bad performance because of multiple micro applications requesting the same data with one request each.

The handling of non-functional requirements was brought up by one interviewee as being extra important in a micro frontends project. As the interviewee argued, having several isolated teams working on different applications that are supposed to be composed into one application puts high demands on requirements regarding for example browser support being clear and well-defined. This differs from microservices in backend development, where the micro services can be run in different environments and do not have to be integrated into one single application. It could therefore be relevant to see it as a potential risk that teams develop micro applications that do not work in the same environments. On a similar topic, one interviewee saw problems in how the idea that teams should not have to communicate would affect the experience of the users of the application since the micro applications are developed by different teams. Examples described were that the UI design and the structure of error messages need to be consistent throughout the application. This is not relevant for microservices in general but is a problem solely relevant for micro frontends since it includes the frontend of an application. Further, two interviewees addressed that the micro applications potentially could end up being of varying code quality if requirements were not clear across the teams, which could also be treated as a risk that should be handled in micro frontends projects.

The concern two interviewees had about knowledge about a micro application disappearing from a company if the application does not require work during a long time could also be considered a risk in micro frontends projects. This is not unique for micro frontends, but relevant for all microservice architectures. However, as one of the interviewees explained,

since things move extra fast in the web community and techniques quickly become out of fashion to use, it is a risk extra important to consider in a micro frontends project for a web application.

One thing that interviewees repeatedly came back to was that the benefits of using micro frontends, or microservice architecture in general, only can be exploited if the services are isolated to a very far extent, meaning that there should be as little dependencies between them as possible. As mentioned in the results, one interviewee actually stated that the most severe risk to consider when starting a micro frontends project should be that the project should not be implemented with a microservice architecture at all simply because the software could not be divided into isolated, small applications. The interviewee specifically stated that implementing microservice architecture when it is not needed introduces unnecessary complexity in the project because of the extensive deployment configuration. This contradicts the idea of Myers [13] that the micro frontends architecture might be simpler than other web architectures, which could make MFAs easier to reason about and manage. Several of the other interviewees did state that extensive overhead in the form of deployment configuration in microservice architecture is a fact, which supports the first mentioned interviewee's view rather than Myers's.

The fact that absence of dependencies is emphasised to be of utmost importance by many of the interviewees implies that micro frontends projects require a lot of knowledge about a project before it has started. In software development, it is natural that some requirements on the implementation, there amongst required dependencies are discovered as the project evolves. In a project that does not adopt micro frontends, the architecture *can* be adjusted as the need for the new dependencies are discovered, even if it is not *desirable*, but in a micro frontends project, the new dependencies imply that the architecture is no longer a clean microservice architecture. This puts high demands on the creation of the architecture at the beginning of a micro frontends project.

At the same time as interviewees expressing that absence of dependencies are critical, some of the thoughts of the interviewees imply that there are some difficulties regarding dependencies that are specific for micro frontends architecture and not relevant for microservices in backend development. For example, if the user should be able to log in to the application, there are probably more than one micro application in the frontend that need the user credentials. Another example is that the same data often is used on different places in the UI, which was also discussed as a possible problem in *6.1.1 Implementations*.

To summarise, some of the thoughts and opinions from the interviews support some of the claims made about benefits of micro frontends in the studied literature presented in *3.2 Micro Frontends*. The interviewees also expressed ideas of possible strengths of micro frontends that has not been found in the literature. Furthermore, several challenges and potential risks to handle in micro frontends projects were also brought up. This gives a more nuanced picture of micro frontends than the one painted in the literature.

It can be discussed what the results from the interviews say about the implemented prototypes. As described, two of the micro applications of the MFA are dependent on the same data, and the menu micro application is dependent on the other two micro applications. At the same time, the interviewees stressed that dependencies in a microservice project should be kept to a minimum for it to have any advantages compared to a monolithic solution. This speaks against the MFA prototype being a suitable solution. One interviewee specifically described it as inappropriate to implement a menu as a micro application, partly because it would be dependent on the other applications, but also because of its simplicity in functionality. Other interviewees also thought that the micro applications in a micro frontends project should be rather complex with a lot of responsibilities. This cannot be said about the MFA prototype, which consists of three rather simple applications with few features. Further,

several interviewees thought that micro frontends should be used in large project, too large to be handled by one team. Neither this can be said about the MFA prototype.

6.1.3 Mathematical Modelling

The results from using the SQMMA model on the prototypes indicated that the changeability was better for the SPA, the stability for the MFA, and finally, that the modifiability was better for the SPA. However, it is hard to tell if the difference in modifiability should be considered to be large. In Chawla and Chhabra's work [48], the largest difference between the normalised modifiability values of the studied source codes was 0.04, while the difference between the values of the SPA and the MFA was 0.314. This could indicate that the difference between the SPA and the MFA is large. On the other hand, Chawla and Chhabra do not demonstrate the modifiability scores before they were normalised, and it can be misleading to compare the calculated scores of this study with scores of Chawla and Chhabra's study. The reason for this is that the modifiability scores of the SPA and the MFA are relatively small, meaning that relatively small absolute differences give a large percental difference compared to if the values were relatively large. Therefore, if the modifiability scores of Chawla and Chhabra's work are very large compared to the scores of the SPA and the MFA, the absolute difference between Chawla and Chhabra's source code versions could be significantly larger than the absolute difference between the SPA and the MFA. In that case, the comparison of the normalised values is pretty useless. We can therefore not say if the difference between the modifiability scores of the SPA and the MFA is large. We can only say that there is a difference, and that the SPA had a better modifiability score.

Considering which code metrics the modifiability values are based on, the values could indicate that the MFA prototype consists of more code of larger complexity than the SPA prototype, which is accurate. This goes in line with what some of the interviewees said about the usage of micro services resulting in overhead in the form of setup code and the architecture being rather complex. This is also something that is widely known about micro service architecture. One could argue that the SQMMA model of modifiability is not very nuanced and only represents a small fraction of what modifiability is. For example, the model does not take into account that having separate deploy flows for every module in an application could facilitate performing modifications in it. Therefore, the model confirms what the interviewees said about overhead and complexity in micro service architecture, but the interviews provided additional information about modifiability in micro frontends that the model cannot express.

6.2 Method

In this section, the different parts of the method of this report are discussed. Insufficiencies and possible effects on the results of the method are debated. Specifically, both threats and decisions made with respect to validity and reliability are addressed. First, definitions of validity and reliability are provided, followed by a definition replicability and how it has been taken into account when writing the method. Then, subsections of different parts of the method are included in which they are discussed.

Generally when talking about validity and reliability in scientific research, the definitions provided by Åberg in [36] are meant. That is, that reliability corresponds to whether the same results can be expected to be obtained if the same study was to be repeated, and that validity corresponds to whether the performed measures actually measure what one thinks is being measured. In qualitative research however, a different definition of reliability should be used, as described by Collingridge and Gantt [74]. They state that in qualitative research, such as semi-structured interviews, a study is considered reliable if the methods used are accepted by the research community as legitimate. Because of this, the subsection about the semi-

structured interviews below will discuss reliability with regards to the definition provided by Collingridge and Gantt. Further, Collingridge and Gantt state that validity is considered the same thing in qualitative research as in quantitative.

When it comes to replicability, Åberg [36] defines a method as replicable if another person reading the report than the author can understand exactly how to execute the method and obtain the same results. In qualitative studies, it is reasonable that the results differ even if the same method is used [36]. To obtain high replicability in this report, the method has been written with as much detail as possible. For example, tools used that could affect the results have been demonstrated, the implemented analysis tool and the interview guide and tests have been included as appendices, and the used code metrics have been described in detail. To validate if the method is easy to understand and follow, three students in the area of computer science read the report and gave feedback on whether the method needed refinement and clarification.

6.2.1 Implementations

The implemented prototypes would probably not be implemented in exactly the same way if another developer would have implemented them. In the same manner, using other tools than React and single-spa would result in other prototypes. This does not necessarily make the results from the implementation unreliable because it is only reasonable that this is the case, but it is worth mentioning.

The developer of the prototypes, that is, the author of this report, also wrote the analysis program and the belonging unit tests and performed the exploratory testing of the prototypes. The developer could be unconsciously biased to the implementations advantages and not analyse and test the implementations thoroughly enough, which can be seen as a threat to reliability. This threat could have been mitigated by letting other developers implement the analysis program, or at least implement the unit tests and perform the exploratory testing. Unfortunately, there was not enough time or resources in the form of developers to do this. An attempt has been made to at least be transparent about what the analysis program and the unit tests contained by including them in the report as appendices. This gives the reader a chance to form an opinion of their own about whether the analysis program and unit tests were implemented in a suitable manner.

6.2.2 Semi-structured Interviews

As in many interview-based methods, the number of samples was limited. The selected interviewees were all from the same company, which means that they could have experiences that are specific for that company, which in turn could mean that the results from the interviews are not very general. It should however be mentioned that the company at least is relatively large, and that the interviewees did not all belong to the same team, or even work in the same cities. Nevertheless, both these factors are threats to the validity of the results and should be taken into consideration when analysing them.

Another threat to validity is the fact that none of the interviewees had actual experience of working with micro frontends, even if most of them had worked with microservices. A lot of the results from the interviews are speculations from the interviewees. It would however been hard to find people willing to participate that have experience of micro frontends, since it is not a widely adopted concept yet.

As described earlier, a qualitative study is considered reliable if methods accepted by the research community as legitimate are used. In this study, semi-structured interviews were performed, which is a widely accepted research method in various fields of research, described by Gray [75], Hove and Anda [29], and Seaman [35], and used by amongst other Haselböck

et al. [33] and Groher and Weinreich [34]. Guidelines for conducting the interviews provided by Seaman were considered, there amongst the suggestion to use an interview guide, which was also used by Haselböck et al. [33] and Groher and Weinreich [34]. Further, as done by Haselböck et al. [33] and Groher and Weinreich [34], the data analysis procedure suggested by Mayring [37] was used. Because all these widely accepted and used methods for executing semi-structured interviews were used in this study, it should be considered reliable.

6.2.3 Mathematical Modelling

There exist several threats to validity with the usage of the SQMMA model in this report. Firstly, the SQMMA model was developed for object-oriented code, and even though React in several ways adopts object-oriented concepts, it is not object-oriented in the same sense as plain JavaScript. It is therefore highly relevant to question if the adjustments made to be able to use it on React yields representative results. Secondly, because Chawla and Chhabra [48] do not provide definitions of the required metrics, assumptions had to be made about these, and these assumptions may have resulted in that the metrics being used were far from what Chawla and Chhabra intended. If other models of modifiability would have been found than Chawla and Chhabra's or Dayanandan and Kalimuthu's, these would definitely have been considered instead, but this was not the case. The results from the mathematical modelling should therefore not be considered as definite truths about the modifiability of the prototypes, but rather as possible indications.

6.3 The Work in a Wider Context

An aspect worth considering when adopting the micro frontends concept that has not been addressed yet, but is highly relevant when talking about it in a wider context, is that of environmental impact. In 2018, it was estimated that global data centers account for 1% of worldwide electricity consumption [76]. Because the micro frontends architecture implies more overhead in the form of for example project setup and deployment compared to a monolithic architecture, it is only reasonable to assume that the micro frontends architecture requires more computing power. If micro frontends would become the new standard of implementing web applications, the global electricity consumption could be affected significantly.

7 Conclusion

In this chapter, the purpose of this study is revisited and the generalisability of the conclusions that will be presented are discussed. This is followed by conclusions in the form of answers to the research questions. Lastly, ideas for continued work are described.

The purpose of this study was to explore the benefits and challenges of micro frontends, partly with focus on modifiability. Further, it aimed at investigating when micro frontends should be used, and specifically if it should be used when rebuilding the application described in *2.2 Existing Solution*. When it comes to the generalisability of the answers to the research questions, the conclusions made based on the interviews can be valuable for everyone thinking of starting a micro frontends project, even though the sampling of interviewees was limited. The provided answers should not be considered to be general truths, but rather aspects to contemplate. The conclusions made based on the SQMMA results are specific to the implemented prototypes and cannot be considered to be general for SPAs and micro frontends applications. This, because the results depend heavily on how the prototypes were developed and which tools were used.

7.1 Research Questions

What are the effects of adopting the micro frontends concept in a web application project?

a) **What do practitioners believe are the positive effects on the development of the software as well as the software itself?**

The following positive effects of adopting micro frontends were brought up by the interviewees:

- Teams become experts on their owned feature.
- Responsibilities of each team are clear.
- Small risk of misunderstandings between frontend and backend developers.
- Having full stack teams could result in more varied work tasks, which is generally appreciated by developers.

- Allows for small releases of certain parts of an application.
- Optimisation of technique choice is enabled.
- Most parts of an application can still be running even if one micro application breaks.
- Independent, isolated, and well-defined components makes testing easier.
- Testing many small services can be smoother than testing one large monolith.

b) **What risks should be considered when starting a project using the micro frontends concept according to practitioners?**

The following potential risks to consider when starting a micro frontends project were brought up by the interviewees:

- Teams may run out of tasks.
- If allowing team members to jump between teams, it could take time before they become productive since different teams can use different techniques and design patterns.
- Knowledge about a micro application might disappear from the company if it does not require work during a long time.
- Performance issues may become a problem if using multiple frameworks in frontend.
- The code quality of different micro applications in a micro frontends project can vary.
- The user experience can be affected negatively by letting different teams implement different parts of the frontend, for example by the UI design not being consistent.
- It can be discovered in a late stage of the project that there is a need for dependencies between the micro applications, meaning that either the architecture was designed inconveniently, or that the architecture should not follow the micro frontends pattern at all. This would destroy all benefits that could be obtained from micro frontends, and only introduce more problems.

How is the modifiability of an application affected by adopting the micro frontends concept?

a) **What do practitioners believe are the implications of implementing a web application using the micro frontends concept with regards to modifiability?**

The following potential effects on modifiability could be derived from the answers from the interviews:

- Making incremental upgrades of an application, which is enabled by micro frontends, is smoother and easier than making one big upgrade.
- Smaller teams, which micro frontends naturally allows for, facilitates further development.
- Isolated code bases running side by side, with no dependencies, makes it easy for developers to perform changes in their relatively small codebase compared to in a large monolith, were one small part of the monolith can break the entire system.
- It can be smoother to add a new feature as an isolated service than integrating it as a part of an existing application.

- Smaller codebases, deploy pipelines and version control repositories that do not get pushed to by a large number of developers facilitate modifications.
- If a micro application has not been worked on for a while and the competence for working in it has disappeared from the company, performing changes in it will be inconvenient.

b) **Can the claim that the modifiability of a micro frontends application is better than that of an SPA be verified with a quantitative modifiability model using code metrics?**

No, it cannot. According to the SQMMA model, the SPA prototype is more modifiable than the MFA prototype. This should however be taken with a grain of salt, because of the aspects discussed in *6.1.3 Mathematical Modelling* and *6.2.3 Mathematical Modelling*.

When should the micro frontends concept be used?

a) **In which type of projects do practitioners believe it can be valuable to adopt the micro frontends concept, and when should it not be used?**

Micro frontends should only be used when building large applications, so large that they are too big to handle by one team. On a similar note, it is valuable if a web application should do many different things and have many unrelated responsibilities. In that case, the concept can be useful when a company wants to outsource work on an application to different consultancy firms and they should not have to cooperate. For smaller applications, it should not be used. Further, it should be possible to divide the application into more than a few micro applications, otherwise it would be unnecessary to distribute the applications across different teams and the overhead in the form of deployment configuration would be worthless and only introduce excessive complexity. It is however worth considering using micro frontends, even if the application will start out small, if there is a chance that the application will grow bigger in the future. An additional thing to acknowledge when deciding if micro frontends should be used in a project is feature complexity. If an application has to be divided into micro applications providing relatively simple features that perform little work and have few responsibilities, it would be wasteful to dedicate a team to that micro application and, yet again, the architecture would only introduce unnecessary complexity. One of the most important factors to acknowledge when considering using micro frontends is whether it is possible to divide the application into micro applications that are almost completely isolated and independent of each other. Otherwise, the benefits of using the concept are lost. Further, micro frontends should not be used if there is a tight deadline or a small budget because of the implied overhead.

b) **Is it suitable to use for the application presented in *2.2 Existing Solution*?**

The short answer is no, micro frontends should not be used when rebuilding the application presented in *2.2 Existing Solution*. This answer is simply derived from the fact that the application is small, and the features are simple. The MFA prototype was divided in a manner that meant that the micro applications were not independent and isolated, and it is hard to imagine how it could have been divided in a different way.

7.2 Future Work

If there would be more time available to conduct this work, effort would be put on interviewing additional practitioners. Specifically, practitioners from different companies would be selected. This would make the results more general, regardless of if the new results only

confirm the existing results or add new ones. For the curious reader, Mezzalira, whose talk [7] from 2020 has been cited earlier in this report, published a new scientific report [77] on current knowledge about micro frontends in March this year together with two other authors. Possibly, the report contains information relevant for future work in this research area.

Bibliography

[1] R. Kulesza, M. F. de Sousa, M. L. M. de Araújo, C. P. de Araújo, and A. M. Filho, "Evolution of Web Systems Architecture: A Roadmap," in *Special Topics in Multimedia, IoT and Web Technologies*, V. Roesler, E. Barrére, and R. Willrich, Eds. Cham, Switzerland: Springer, 2020, pp. 3–21, ISBN: 978-3-030-35101-4.

[2] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, Jan. 2015. DOI: 10.1109/MS.2015.11.

[3] C. Richardson. (Unknown). "Pattern: Monolithic Architecture," [Online]. Available: https://microservices.io/patterns/monolithic.html. 2021-02-08.

[4] S. Newman, "Microservices," in *Building Microservices*, M. Loukides and B. MacDonald, Eds. California, United States: O'Reilly Media, 2015, pp. 1–12, ISBN: 978-1-491-95035-7.

[5] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges When Moving from Monolith to Microservice Architecture," in *Current Trends in Web Engineering*, vol. 10544, Rome, Italy, 2017, pp. 32–47. DOI: 10.1007/978-3-319-74433-9_3.

[6] C. Jackson. (2019). "Micro Frontends," [Online]. Available: https://martinfowler.com/articles/micro-frontends.html. 2021-02-08.

[7] L. Mezzalira, "Lessons from dazn: Scaling your project with micro-frontends," London, England: QCon, 2020, [Online]. Available: https://www.infoq.com/presentations/dazn-microfrontend/.

[8] M. Geers, "What are micro frontends?" In *Micro Frontends in Action*, T. Louvar, L. Lazaris, I. Martinovíc, D. S. Hiam, and B. Berg, Eds. New York, United States: Manning Publications Co., 2020, pp. 3–22, ISBN: 9781617296871.

[9] T. Betts and C. Humble, "Software Architecture and Design InfoQ Trends Report – April 2020," InfoQ, 2020. [Online]. Available: https://www.infoq.com/articles/architecture-trends-2020/.

[10] *AngularJS*, https://angularjs.org/, 2021-04-09.

[11] S. Smith, "Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure," Microsoft Corporation, Guide, version 5.0, 2021.

[12] M. Geers. (Unknown). "What are Micro Frontends," [Online]. Available: https://micro-frontends.org/. 2021-02-19.

[13] B. Myers. (Year unknown). "The Strengths and Benefits of Micro Frontends," [Online]. Available: https://www.toptal.com/front-end/micro-frontends-strengths-benefits. 2021-04-12.

[14] C. Yang, C. Liu, and Z. Su, "Research and Application of Micro Frontends," *IOP Conference Series: Materials Science and Engineerin*, vol. 490, no. 6, Apr. 2019. DOI: 10.1088/1757-899X/490/6/062082.

[15] *Mooa*, https://phodal.github.io/mooa/, 2021-04-12.

[16] A. Pavlenko, N. Askarbekuly, S. Megha, and M. Mazzara, "Micro-frontends: Application of microservices to web front-ends," *Journal of Internet Services and Information Security (JISIS)*, vol. 10, no. 2, pp. 49–66, May 2020. DOI: 10.22667/JISIS.2020.05.31.049.

[17] M. Shakil and A. Zoitl, "Towards a Modular Architecture for Industrial HMIs," in *Proc. 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vienna, Austria, 2020, pp. 1267–1270. DOI: 10.1109/ETFA46521.2020.9212011.

[18] "Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models," in *ISO/IEC 25010:2011*, ISO/IEC standard, Mar. 2011.

[19] "Software engineering – Product quality – Part 1: Quality Model," in *ISO/IEC 25010:2011*, ISO/IEC standard, Jun. 2001.

[20] ARiSA. (2009). "Changeability," [Online]. Available: http://www.arisa.se/compendium/node64.html. 2021-04-08.

[21] I. Padayachee, P. Kotzé, and A. V. der Merwe, "ISO 9126 external systems quality characteristics, sub-characteristics and domain specific criteria for evaluating e-Learning systems," The Southern African Computer Lecturers' Association, University of Pretoria, 2010.

[22] ARiSA. (2009). "Stability," [Online]. Available: http://www.arisa.se/compendium/node67.html. 2021-04-08.

[23] J. Bosch, "Design and Use of Software Architectures," in *Proc. Technology of Object-Oriented Languages and Systems. TOOLS 29 (Cat. No.PR00275)*, Nancy, France, 1999. DOI: 10.1109/TOOLS.1999.779093.

[24] C. D. Rosso, "Continuous evaluation through software architecture: a case study," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 5, pp. 351–383, Sep. 2006. DOI: 10.1002/smr.337.

[25] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for Architecture Evaluation," Software Engineering Institute, Carnegie Mellon, Technical report, Aug. 2000.

[26] R. Kazman, G. D. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," *IEEE Software*, vol. 13, no. 6, pp. 47–55, Nov. 1996. DOI: 10.1109/52.542294.

[27] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *Proc. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*, Victoria, BC, Canada, 1998, pp. 308–317. DOI: 10.1109/ICSR.1998.685756.

[28] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976. DOI: 10.1109/TSE.1976.233837.

[29] S. E. Hove and B. Anda, "Experiences from Conducting Semi-Structured Interviews in Empirical Software Engineering Research," in *11th IEEE International Software Metrics Symposium (METRICS'05)*, Como, Italy, 2005. DOI: 10.1109/METRICS.2005.24.

[30] D. Gray, "Interviewing," in *Doing Research in the Business World*, J. Seaman, G. Shields, T. Bedford, and J. Birch, Eds. London, United Kingdom: SAGE Publications, 2017, pp. 405–438, ISBN: 978-1-4739-1567-1.

[31] S. Haselböck, R. Weinreich, and G. Buchgeher, "Decision guidance models for microservices: service discovery and fault tolerance," in *Proc. Fifth European Conference on the Engineering of Computer-Based Systems*, ser. ECBS '17, Larnaca, Cyprus, 2017. DOI: 10.1145/3123779.3123804.

[32] ——, "Decision Models for Microservices: Design Areas, Stakeholders, Use Cases, and Requirements," in *Software Architecture*, A. Lopes and R. de Lemos, Eds. Cham, Switzerland: Springer, 2017, pp. 155–170, ISBN: 978-3-319-65830-8. DOI: 10.1007/978-3-319-65831-5_11.

[33] ——, "An Expert Interview Study on Areas of Microservice Design," in *Proc. 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*, Paris, France, 2018, pp. 137–144. DOI: 10.1109/SOCA.2018.00028.

[34] I. Groher and R. Weinreich, "An Interview Study on Sustainability Concerns in Software Development Projects," in *Proc. 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Vienna, Austria, 2017, pp. 350–358. DOI: 10.1109/SEAA.2017.70.

[35] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, Jul. 1999. DOI: 10.1109/32.799955.

[36] J. Åberg. (2015). "Anvisning för exjobbsrapporter," [Online]. Available: https://www.ida.liu.se/edu/ugrad/thesis/templates/Exjobb_anvisning_150313.pdf. 2021-05-17.

[37] P. Mayring, "Qualitative Content Analysis," *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, vol. 1, no. 2, Jun. 2000. DOI: 10.17169/fqs-1.2.1089.

[38] "Systems and software engineering — Vocabulary," in *ISO/IEC/IEEE 24765:2017(E)*, ISO/IEC/IEEE standard, Sep. 2017. DOI: 10.1109/IEEESTD.2017.8016712.

[39] M. Lanza and R. Marinescu, "Facts on measurements and visualization," in *Object-Oriented Metrics in Practice*, R. Gerstner, Ed. Berlin, Germany: Springer, 2006, pp. 11–22, ISBN: 978-3-540-24429-5.

[40] J. Bansiya and C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4–17, Jan. 2002. DOI: 10.1109/32.979986.

[41] J. Bansiya, L. Etzkorn, C. G. Davis, and W. Li, "A Class Cohesion Metric For Object-Oriented Designs," *Journal of Object-Oriented Programming*, vol. 11, no. 8, pp. 47–52, Jan. 1999.

[42] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, Nov. 2005. DOI: 10.1109/TSE.2005.112.

[43] informIT. (2010). "Using Metrics to Find Out if Your Code Base Will Stand the Test of Time," [Online]. Available: https://www.informit.com/articles/article.aspx?p=1561879&seqNum=3. 2021-04-08.

[44] S. R. Chidamber and C. F. Kemerer, "A Metrics Suit for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994. DOI: 10.1109/32.295895.

[45] F. B. e Abreu and R. Carapuça, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process," in *Proc. Fourth international conference on software quality*, McLean, VA, USA, 1994.

[46] R. Harrison, S. Counsell, and R. Nithi, "An Overview of Object-Oriented Design Metrics," in *Proc. Eighth IEEE International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering*, London, UK, 1997, pp. 230–235. DOI: 10.1109/STEP.1997.615494.

[47] M. F. S. Oliveira, R. M. Redin, L. Carro, L. da Cunha Lamb, and F. R. Wagner, "Software quality metrics and their impact on embedded software," in *2008 Fifth International Workshop on Model-based Methodologies for Pervasive and Embedded Software*, Budapest, Hungary, 2008, pp. 68–77. DOI: 10.1109/MOMPES.2008.11.

[48] M. K. Chawla and I. Chhabra, "SQMMA: Software Quality Model for Maintainability Analysis," in *Proc. 8th Annual ACM India Conference*, ser. Compute '15, Ghaziabad, India, 2015, pp. 9–17. DOI: 10.1145/2835043.2835062.

[49] U. Dayanandan and V. Kalimuthu, "A Fuzzy Analytical Hierarchy Process (FAHP) Based Software Quality," *International Journal of Intelligent Engineering Systems*, vol. 11, no. 4, pp. 88–96, Aug. 2018. DOI: 10.22266/ijies2018.0831.09.

[50] I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos, "The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation," in *Open Source Development, Communities and Quality*, ser. IFIP – The International Federation for Information Processing, vol. 275, Milano, Italy, 2008, pp. 237–248. DOI: 10.1007/978-0-387-09684-1_19.

[51] T. L. Saaty and L. G. Vargas, *Models, Methods, Concepts Applications of the Analytic Hierarchy Process*, 2nd ed., F. S. Hillier, Ed., ser. International Series in Operations Research Management Science. New York, United States: Springer, 2012, vol. 175, ISBN: 978-1-4614-3596-9. DOI: 10.1007/978-1-4614-3597-6.

[52] P. Oman and J. Hagemeister, "Metrics for Assessing a Software System's Maintainability," in *Proc. Conference on Software Maintenance 1992*, Orlando, United States, 1992, pp. 337–344. DOI: 10.1109/ICSM.1992.242525.

[53] T. Kuipers and J. Visser, "Maintainability index revisited – position paper," Jan. 2007.

[54] A. van Deursen. (2014). "Think Twice Before Using the "Maintainability Index"," [Online]. Available: https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/. 2021-04-08.

[55] Facebook, *React*, https://reactjs.org/, 2021-04-06.

[56] S. Greif and R. Benitte. (2019). "State Of JS 2019 – Demographics," [Online]. Available: https://2019.stateofjs.com/demographics/. 2021-04-12.

[57] ——, (2019). "State Of JS 2019 – Front End Frameworks," [Online]. Available: https://2019.stateofjs.com/front-end-frameworks/. 2021-04-12.

[58] Facebook, *Components and Props*, https://reactjs.org/docs/components-and-props.html, 2021-04-12.

[59] ——, *Introducing Hooks*, https://reactjs.org/docs/hooks-intro.html, 2021-04-12.

[60] ——, *Composition vs Inheritance*, https://reactjs.org/docs/composition-vs-inheritance.html, 2021-04-12.

[61] *single-spa*, https://single-spa.js.org/docs/getting-started-overview, 2021-04-06.

[62] C. Kaner. (2008). "A Tutorial in Exploratory Testing," [Online]. Available: https://www.kaner.com/pdfs/QAIExploring.pdf. 2021-06-19.

[63] *Material-UI*, https://material-ui.com/, 2021-04-06.

[64] *npx*, https://docs.npmjs.com/cli/v7/commands/npx, 2021-04-06.

[65] *create-react-app*, https://github.com/facebook/create-react-app, 2021-04-06.

[66] *create-single-spa*, https://single-spa.js.org/docs/create-single-spa/, 2021-04-06.

[67] *glob – Unix style pathname pattern expansion*, https://docs.python.org/3/library/glob.html, 2021-04-08.

[68] *os – Miscellaneous operating system interfaces*, https://docs.python.org/3/library/os.html, 2021-04-08.

[69] *re – Regular expression operations*, https://docs.python.org/3/library/re.html, 2021-04-08.

[70] *NumPy*, https://numpy.org/, 2021-04-08.

[71] *unittest — Unit testing framework*, https://docs.python.org/3/library/unittest.html, 2021-05-03.

[72] *parameterized*, https://github.com/wolever/parameterized, 2021-05-03.

[73] *npm*, https://docs.npmjs.com/cli/v7/commands/npm, 2021-04-20.

[74] D. S. Collingridge and E. E. Gantt, "The quality of qualitative research," *American Journal of Medical Quality*, vol. 23, no. 5, pp. 389–395, Sep. 2008. DOI: 10.1177/1062860608320646.

[75] J. Seaman, G. Shields, T. Bedford, and J. Birch, Eds., *Doing Research in the Business World*. London, United Kingdom: SAGE Publications, 2017, ISBN: 978-1-4739-1567-1.

[76] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science Magazine*, vol. 367, no. 6481, pp. 984–986, Feb. 2020. DOI: 10.1126/science.aba3758.

[77] S. Peltonen, L. Mezzalira, and D. Taibi, "Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review," *Information and Software Technology*, vol. 136, no. 106571, Mar. 2021. DOI: 10.1016/j.infsof.2021.106571.

A Interview Guide

Demographic Questions

- What is your current role at your company?
- How many years have you been working within the software field?
- Do you have experience with working with micro services?
  If yes:
  - What kind of experience?
- Do you have experience in frontend development?

If experience with micro services:

Questions on Micro Services

- What benefits do you think there are with working with micro services?
- What challenges do you think there are with working with micro services?
- What are your thoughts on maintenance of micro services?

Questions on Micro Frontends

- What are your spontaneous thoughts on micro frontends?
- What are your thoughts on deploying frontend modules separately?
- What are your thoughts on the team structure ideas of micro frontends?
  - Do you see any benefits?
  - Do you see any challenges?
- What consequences do you think there are to using full stack teams instead of teams split on frontend and backend?

- One of the supposed benefits of using micro frontends is that there is no need for communication across teams. What are your thoughts on that?
- What effects do you think there are on the testing of an application when it is implemented with the micro frontends concept?
- Are there any specific cases when it is a good idea to adopt the micro frontends concept?
- Are there any specific cases when it is not a good idea?
- Can you see any risks you should consider when starting a project involving micro frontends?

B Analysis Program Functions

In this section, code implemented for calculating the needed metrics is provided. First, each function used for obtaining a code metric is presented together with the help functions uniquely used by that function, if any. Then, help functions used by several of the code metric functions are provided.

```
from help_functions import get_functions, update_current_depth
import re

def calculate_AMS(file_path):

  functions = get_functions(file_path,
                            include_component_function = False)

  if len(functions) == 0:
    return 0

  total_LLOC = 0
  for function in functions:
    current_depth = 0

    for i in range(1, len(function) - 1):
      current_depth = update_current_depth(function[i],
                                           current_depth)

      if not (len(function[i].strip()) < 3 or
              re.match(' *\n', function[i]) or
              re.findall(' *\/\/ .*', function[i])):
        total_LLOC += 1

  return total_LLOC / len(functions)
```

Listing B.1: Function `calculate_AMS`, which returns the *AverageMethodSize* of a file.

```
import glob, os, re

def get_efferent_coupling(file_path):

```

```
    with open(file_path, 'r') as f:
      s = f.read()
      pattern = 'import .*\'\.'
      matches = re.findall(pattern, s)

      return len(matches)


def get_afferent_coupling(project_repository, repository,
                          filename, search_repository):

    pattern = 'import .* from \''
    # If same directory:
    if (repository == search_repository):
      pattern += '.'
    # If one directory level down:
    elif (search_repository == ''):
      pattern += '.' + repository
    # If same directory level:
    else:
      pattern += '..' + repository
    pattern += '/' + filename.replace('.js', '') + '\''

    afferent_coupling = 0
    search_path = '..\\' + project_repository + '\\src' + \
                  search_repository + '/*.js'
    search_path = search_path.replace('/', '\\')

    for search_in in glob.glob(search_path):
      search_in = search_in.replace('/', '\\')

      with open(os.path.join(os.getcwd(), search_in), 'r') as f:
          s = f.read()
          matches = re.findall(pattern, s)
          imports = len(matches)

          if (imports > 0):
            afferent_coupling += 1

    return afferent_coupling


def calculate_CBO(file_path, project_repository,
                  search_respositories):

    EC = get_efferent_coupling(file_path)
    AC = 0
    file_path_array = file_path.split(sep='\\')
    filename = file_path_array[-1]

    repository = ''
    if file_path_array[-2] != 'src':
      repository = '/' + file_path_array[-2]

    for search_repository in search_respositories:
      search_repository = search_repository.replace('src\\', '/')
    [:-1]
      AC += get_afferent_coupling(project_repository, repository,
                                  filename, search_repository)

    return EC + AC
```

Listing B.2: Function `calculate_CBO`, which returns the *CouplingBetweenObjects* of a file together with the help functions `get_efferent_coupling` and `get_afferent_coupling`.

```python
from help_functions import get_functions, update_current_depth
import numpy as np, os, re

def get_component_states(file_path):

  with open(os.path.join(os.getcwd(), file_path), 'r') as file:
    pattern = 'const \[.*, set.*\] = useState\('
    content = file.read()
    matches = re.findall(pattern, content)
    states = []

    for match in matches:
      states.append(match.split('[')[1].split(',')[0])

    return states

  return []


def get_sets_of_used_states(functions, states):

  used_states_sets = []
  for function in functions:
    used_states = []

    for state in states:
      for row in function:
        if (re.search(state, row) or
            re.search(state[0].capitalize() + state[1:], row)):
          used_states.append(state)
          break

    used_states_sets.append(used_states)

  return used_states_sets


def get_intersections(sets):

  # Number of null intersections
  NNI = 0
  # Number of nonempty intersections
  NNEI = 0

  for i in range(len(sets)):
    for j in range(i + 1, len(sets)):
      if len(np.intersect1d(sets[i], sets[j])) > 0:
        NNEI += 1
      else:
        NNI += 1

  return NNI, NNEI


def calculate_LCOM(file_path):

  functions = get_functions(file_path,
                            include_component_function = False)
  states = get_component_states(file_path)

  used_states_sets = get_sets_of_used_states(functions, states)
  NNI, NNEI = get_intersections(used_states_sets)

  LCOM = NNI - NNEI

```

```
66    return LCOM if LCOM > 0 else 0
```

Listing B.3: Function `calculate_LCOM`, which returns the *LackOfCohesion* of a file together with the help functions `get_component_states`, `get_sets_of_used_states`, and `get_intersections`.

```
1  from help_functions import get_functions, update_current_depth
2
3  def calculate_NBD(file_path):
4
5    functions = get_functions(file_path,
6                              include_component_function = False)
7
8    total_NBD = 0
9    for function in functions:
10     max_depth = 0
11     current_depth = 0
12
13     function.pop(0)
14     for line in function:
15       current_depth = update_current_depth(line, current_depth)
16
17       if (current_depth > max_depth):
18         max_depth = current_depth
19
20     total_NBD += max_depth
21
22   return total_NBD / len(functions) if total_NBD != 0 else 0
```

Listing B.4: Function `calculate_NBD`, which returns the *NestedBlockDepth* of a file.

```
1  from help_functions import get_functions
2  import os, re
3
4  def get_function_calls(function_content):
5
6    function_calls = []
7
8    function_content.pop(0)
9    for row in function_content:
10     matches = re.findall('[^ ]\(.*?', row)
11
12     for function_call in matches:
13       function_calls.append(function_call)
14
15   return function_calls
16
17
18 def calculate_RFC(file_path):
19
20   R_all = []
21   M = get_functions(file_path,
22                     include_component_function = False)
23
24   for function in M:
25     R_all.append(get_function_calls(function))
26
27   RFC = len(M)
28   for R_i in R_all:
29     RFC += len(R_i)
30
```

```
    return RFC
```

Listing B.5: Function `calculate_RFC`, which returns the *ResponseForClass* of a file together with the help functions `get_function_calls`.

```
import os, re

def update_current_depth(line, current_depth):

  opening_braces = re.findall('{', line)
  closing_braces = re.findall('}', line)

  return current_depth + len(opening_braces) - len(closing_braces)


def get_functions(file_path, include_component_function):

  with open(os.path.join(os.getcwd(), file_path), 'r') as file:

    if (include_component_function):
      pattern = ('((const .* \= \(.*\) => {|'
                 'function .*\(.*\) {)|'
                 '.*\(.*\) => {)')
    else:
      pattern = ('(const .* \= \(.*\) => {|'
                 '.*\(.*\) => {)')

    file_content = file.readlines()
    in_function = False
    current_depth = 0
    functions_content = []

    function_lines = []
    for line in range(0, len(file_content)):
      match = re.findall(pattern, file_content[line])

      if match:
        function_lines.append(line)

    for function_line in function_lines:
      current_depth = 0
      start_line = function_line

      for line in range(function_line, len(file_content)):
        current_depth = update_current_depth(file_content[line],
                                             current_depth)

        if (current_depth == 0):
          function_content = file_content[start_line : line + 1]
          functions_content.append(function_content)
          break

    return functions_content

  return []
```

Listing B.6: Help functions `update_current_depth` and `get_functions` used by several of the functions for deriving the needed code metrics.

C Analysis Program Tests

In this section, the unit test program for testing the analysis program is included.

```
from AMS import calculate_AMS
from CBO import calculate_CBO
from LCOM import calculate_LCOM
from NBD import calculate_NBD
from RFC import calculate_RFC
from parameterized import parameterized
import unittest

class AMS(unittest.TestCase):

  data_repository = '..\\test_data\\'

  @parameterized.expand([
    ['activity-functions', 1.333],
    ['App', 0],
    ['Button', 0],
    ['FeatureToggles', 5],
    ['index', 0],
    ['InfoPanel', 0],
    ['NavMenu', 7.666],
    ['NewInfoPost', 1],
    ['Search', 3]
  ])
  def test_AMS(self, name, expected):
    AMS = calculate_AMS(self.__class__.data_repository + \
                        name + '.js')
    self.assertAlmostEqual(AMS, expected, places = 2)


class CBO(unittest.TestCase):

  data_repository = '..\\test_data\\CBO\\src\\'

  @parameterized.expand([
    ['App', '', 10],
    ['AppVersion', 'pages\\', 3],
    ['Button', 'components\\', 1],
```

```
      ['NewInfoPost', 'components\\', 4],
      ['PurpleButton', 'components\\', 3]
    ])
    def test_CBO(self, name, repository, expected):
      CBO = calculate_CBO(self.__class__.data_repository + \
                          repository + name + '.js',
                          'test_data/CBO',
                          ['src\\',
                           'src\\components\\',
                           'src\\pages\\'])
      self.assertAlmostEqual(CBO, expected, places = 2)


  class LCOM(unittest.TestCase):

    data_repository = '..\\test_data\\'

    @parameterized.expand([
      ['activity-functions', 15],
      ['App', 0],
      ['Button', 0],
      ['FeatureToggles', 0],
      ['index', 0],
      ['InfoPanel', 0],
      ['NavMenu', 0],
      ['NewInfoPost', 1],
      ['Search', 0]
    ])
    def test_LCOM(self, name, expected):
      LCOM = calculate_LCOM(self.__class__.data_repository + \
                            name + '.js')
      self.assertAlmostEqual(LCOM, expected, places = 2)


  class NBD(unittest.TestCase):

    data_repository = '..\\test_data\\'

    @parameterized.expand([
      ['activity-functions', 0.166],
      ['App', 0],
      ['Button', 0],
      ['FeatureToggles', 1],
      ['index', 0],
      ['InfoPanel', 0],
      ['NavMenu', 1.333],
      ['NewInfoPost', 0],
      ['Search', 0]
    ])
    def test_NBD(self, name, expected):
      NBD = calculate_NBD(self.__class__.data_repository + \
                          name + '.js')
      self.assertAlmostEqual(NBD, expected, places = 2)


  class RFC(unittest.TestCase):

    data_repository = '..\\test_data\\'

    @parameterized.expand([
      ['activity-functions', 11],
      ['App', 0],
      ['Button', 0],
      ['FeatureToggles', 6],
      ['index', 0],
      ['InfoPanel', 0],
```

```
104       ['NavMenu', 7],
105       ['NewInfoPost', 4],
106       ['Search', 4]
107     ])
108     def test_RFC(self, name, expected):
109       RFC = calculate_RFC(self.__class__.data_repository + \
110                           name + '.js')
111       self.assertAlmostEqual(RFC, expected, places = 6)
112
113
114   if __name__ == "__main__":
115     unittest.main()
```

Listing C.1: Program executing unit tests on functions for deriving needed code metrics in the analysis program.