
How to Make the SOK Fit Web Applications?



Master Thesis

June 26, 2014

Version 1.0

Author:

R. HOVING

r.hoving@students.uu.nl

Master Business Informatics

Utrecht University

Supervisors:

dr. ir. J.M.E.M. VAN DER WERF (Utrecht University)

dr. S.L.R. JANSEN (Utrecht University)

R. DE JONG (AFAS Software B.V.)

Additional Information

Thesis Title	How to Make the SOK Fit Web Applications?
Author	R. Hoving
Student ID	3969290
First Supervisor	dr. ir. J.M.E.M. van der Werf Department of Information and Computing Sciences Utrecht University
Second Supervisor	dr. S.L.R. Jansen Department of Information and Computing Sciences Utrecht University
External Supervisor	R. de Jong Director Architecture & Innovation AFAS Software B.V.

*”There is a single light of science, and
to brighten it anywhere is to brighten
it everywhere.”*

Isaac Asimov

Acknowledgments

This master thesis report was performed as a graduation project for the Business Informatics master program at the Utrecht University. During the entire research project, AFAS Software B.V. provided me with a place to work.

During my research, I was surrounded by many people who have offered me guidance, support, and feedback. I would like to acknowledge these people further. I would like to thank my supervisors from the Utrecht University, Jan Martijn van der Werf and Slinger Jansen. They have provided me with needed feedback and have allocated much of their time to help me improve my thesis. I would like to thank Jan Martijn in special for taking the time to answer my questions and provide me with feedback. Even when he himself had little available time, he still took the time to aid me in my research.

At AFAS Software B.V., my abilities were constantly challenged by my supervisors and colleagues. As a result, my abilities and the thesis itself were greatly improved during the project. I would like to thank Michiel Overeem for answering my questions and helping me on a daily basis. The expertise of Michiel allowed me to greatly improve my knowledge about software, architecture, and programming. Furthermore, I would like to thank Machiel de Graaf for looking after and steering the research. He helped me to prioritize, to keep an overview, and to stay in the right direction. I would like to thank Rolf de Jong for keeping the big picture in mind and reminding me when I steered away from this big picture. Furthermore, Henk van der Schuur has helped me by sharing his knowledge about Software Operation Knowledge and the Software Operation Knowledge framework.

To help me improve my thesis, 8 persons have read my thesis and provided me with feedback. Many thanks for Rolf de Jong, Michiel Overeem, Jan Pieter Guelen, Gerard Nijboer, Henk van der Schuur, Alberto Melendez, Andrei Idu, and Bas Vlug for taking the time to read my thesis and provide me with feedback. Because of them, I could greatly improve the value of my thesis. I would especially like to thank Jan Pieter, Gerard, and Bas for the sharing of knowledge, experiences, pitfalls, and tips.

Abstract

With the constant improvement of computers and devices, the webclient in a client-server architecture can perform large portions of logic that typically resides server-side. When the architect chooses to upload and execute the business logic on the webclient, the webserver requires fewer resources. Fewer resources result in a smaller server landscape. Although this sounds promising, shifting more logic to the webclient increases the required resources on the webclient. Because of the large amount of possible devices, the architect cannot predict the specific device on which the application runs. When the application performs more logic on the webclient, the user's device increasingly influences the user experience.

The characteristics of web application influence the user experience. To provide the architect with information on the user experience we discuss these characteristics. Web applications use AJAX (Asynchronous Javascript And XML) for communication between client and server. A web application uses the Document Object Model (DOM) to change the HTML document of the web page to communicate with the user. Because of the large amount of possible devices, the architect cannot predict the specific device on which the user interacts with the application. However the architect can measure the user experience when the user is interacting with the application. To provide the architect with these measurements, we utilize the Software Operation Knowledge (SOK) framework.

A metric of user experience which the client heavily influences is user-perceived latency. Using the steps of human computer interaction we provide with a formal definition of user-perceived latency (UPL). In UPL we see two types of actors, the user and the system. The architecture cannot influence the user, but can influence the system. Therefore we divided UPL into two components, Human Interaction Time (HIT) and System Response Time (SRT). User experience is an element used to describe the quality of the system. Because SRT is not a property of the application, we see SRT as a metric. Therefore we describe the relation between SRT and the following quality attributes: functional suitability; reliability; performance efficiency; operability; security; compatibility; maintainability; and transferability.

To measure SRT, we instantiate the SOK framework for SRT based measurements. Architectural erosion and architectural degradation result in uncertainties for the architect. Therefore, the architect is unable to predict the effect of changes in the software's architecture on SRT. To cope with architectural erosion and architectural degradation, the instantiated framework includes a simulation step. The architect reproduces SRT in the simulation step so that they can predict the effect on SRT.

We use the adjusted framework to provide a method aimed at gaining knowledge about SRT and the effects of changes in the software's architecture on SRT. The method aids in measuring, improving, and simulating SRT. To improve SRT, we perform data mining techniques and process mining techniques onto the data. The architect uses presentation means to visualize the data and identify improvements in the architecture. After the architect improves the software's architecture and the software, they create and utilize a test environment to obtain an accurate prediction of the effect of these changes on SRT.

To validate this method we performed a case study at AFAS Software B.V.. During the case study, we put the method into practice to measure SRT of Profit Next. The case study increased the awareness of the importance to measure, monitor, and improve SRT. As a result, the architects are currently improving and adding functionalities to the implementations provided by the method.

Table of Contents

1	Introduction	1
1.1	Scientific and Social Relevance	2
1.2	Thesis Outline	2
2	Research Method	3
2.1	Research Questions	3
2.2	Research Method	4
2.3	Challenges	6
2.4	Final Deliverable	7
2.5	Project Planning	7
3	Web Applications and Their Architecture	9
3.1	Running Example	9
3.2	AJAX Web Application Model	10
3.3	Software Operation Knowledge	13
4	User-Perceived Latency	15
4.1	Human Computer Interaction	15
4.2	Characteristics of User-Perceived Latency	17
4.3	System Response Time and Quality Attributes	18
4.4	Providing Information About the System Response Time	20
4.5	Analyzing System Response Time	22
4.6	Instantiated SOK Framework Fit For SRT	22
5	A Method for Improvement of User-Perceived Latency	25
5.1	A Template Method for Improvement of SRT	25
5.2	Implementation and Supporting Software	27
5.3	Identification of Software Operation Knowledge	28
5.4	Acquisition of Software Operation Knowledge	32
5.5	Presentation of Software Operation Knowledge	33
5.6	Utilization of Software Operation Knowledge	35
5.7	System Response Time Simulation	36
6	Case Study Report	39
6.1	Case Company AFAS Software B.V.	39
6.2	Case Study Structure	41
6.3	Data Analysis	51
6.4	Providing SRT Information	58
6.5	Future Work	70
6.6	Method Discussion	71
6.7	Retrospect	72
7	Discussion and Conclusion	75
7.1	Discussion	78
7.2	Future Work	78

Typical web applications in a client-server architecture depend heavily on the webserver when executing business logic [1]. Performing the business logic on the webserver requires resources e.g. processor speed, disk operations, and RAM. Therefore, an increase in the web application's business logic yields in additional resources required on the webserver. When the webserver requires additional resources, costs of the web application increase.

Today's computers and devices are becoming increasingly faster. Therefore, the architect can use the webclient in a web application to process more business logic. When the architect shifts large portions of the logic from the webserver to the webclient, performance and interactivity increase [2]. Because the webserver has to perform less logic, the webserver requires fewer resources. For web applications with large amounts of logic, shifting large amounts of logic from the webserver to the webclient can result in a noticeable decrease in the webserver's required resources. This decrease can result in a smaller server landscape. A smaller server landscape yields in a decrease in cost. Because of the mentioned reasons, organizations choose to shift logic from the webserver to the webclient more often.

When the logic shifts from the server to the client, resources of the client's device have a bigger influence on the user experience. Because of the heterogeneous landscape of devices, the device on which the application runs becomes difficult to predict. Because of this unpredictability, the architect faces the difficult task to balance the logic between client and server. When the architect decides to process more logic on the client, the required resources of the used device increases. However, the architect cannot control or enforce the used device. We define the problem statement as:

In web applications, the architect does not know the device used to perform business logic on the webclient. Therefore, the architect of a web application cannot make a well substantiated choice on where to perform business logic, webclient or webserver. As a result, the user experience in web applications cannot be predicted.

A metric of user experience, which the user's device heavily influences, is user-perceived latency (UPL). We define UPL as the elapsed time between finding the logic the user wants to perform and the evaluation of the response of the application. Measuring UPL on both the client and the server is required when improving the user experience of a web application. We divide UPL into two types of latency, the time the user interacts with the system and the time required for the system to execute the logic and present the details to the user. Because of the architectural perspective of the research, we focus on the time required for the system to execute the logic and presents the results.

The architect can measure the device on which the user interacts with the web application at the moment of use. Therefore we need to measure UPL whilst the user interacts with the application. To provide with these measurements, the research builds on the Software Operation Knowledge (SOK) framework. The SOK framework aids in gathering knowledge of in-the-field performance, quality and usage of software and provides knowledge of in-the-field end user software experience feedback [3]. To measure and improve UPL, the research defines and uses an altered version of the SOK framework. To measure UPL so that the architect can improve UPL, we use the altered framework to base a method on. We consider UPL improved when a reduction in duration of UPL can be obtained e.g. a measured UPL

functionality was 300 milliseconds and after alteration the architect measured the UPL functionality as 200 milliseconds. We define the objective of this research as:

Using Software Operation Knowledge to improve user-perceived latency and predict the effect of changes in the architecture on user-perceived latency in web applications

When the architect alters the architecture of the application, they can use the altered SOK framework to measure the effect on UPL. Because the architect can only obtain SOK from software in production, this approach is far from an ideal situation. As an example, an alteration might result in a bug which increases UPL. The architect needs to measure the effect of architectural changes on UPL prior to putting the software into production. To predict the effect on UPL, the method includes a simulation phase. The simulation phase aids in setting up a simulation environment which an architect can use to make an accurate prediction on UPL.

1.1 Scientific and Social Relevance

One can measure the performance knowledge in the SOK framework through the measurements as described by Putrycz, Woodside and Wu [4], Johnson, Ho, Maximilien, and Williams [5], and Mani and Nagarajan [6]. However, these measurements only apply to desktop applications and service-based software. Another research gap centers around SOK, data mining, and process mining. When put into practice, one can use SOK, data mining, or process mining to gain knowledge about a certain set of data. However, researchers have yet to describe the manner in which one can use these techniques for comparison. For instance when we implement a technique to obtain a set of data. At another point in time we use the same implemented technique to obtain another set of data. Researchers have yet to describe in literature how an architect can use SOK, data mining, or process mining techniques to compare these two or more sets of data and gain knowledge from the comparison. Neither have these techniques been used for the creation of a test environment.

Because of the advantages of asynchronous web applications over synchronous web applications, large companies rapidly adopt asynchronous web applications [7]. Examples are Google Gmail[8], Google Maps, Facebook, Twitter, GitHub, and Flickr [7, 9].

1.2 Thesis Outline

We structured this thesis in 7 chapters. Chapter 2 covers the main and sub research questions. Using the design-science paradigm as described by Hevner, March, Park, and Ram [10], the chapter describes the research approach. The chapter addresses the expected challenges, and the planning of the research project using a Process Delivery Diagram. Chapter 3 provides more information on the definitions related to web applications and their architecture which we use throughout the research. The chapter describes the running example, the term AJAX, the Document Object Model, the different web application models, and SOK together with the SOK framework. To give more information about user-perceived latency, chapter 4 describes the link between human computer interaction and user-perceived latency. The chapter further describes the characteristics of user-perceived latency. The chapter describes the connection between user-perceived latency and the quality attributes of the software product quality model described by the ISO 25010 standard [11]. Furthermore, the chapter provides information about measuring user-perceived latency, analyzing the user-perceived latency, and provides the improved SOK framework fit for measuring and improving user-perceived latency. Chapter 5 covers the method based on the improved framework. It provides tools to identify the metrics of user-perceived latency, to obtain values for the metrics, to analyze these measurements, to define presentation means which the architect uses to improve the architecture, and how to set up a simulation environment. We utilize the method described in chapter 5 during a case study. We describe the results of the case study described in chapter 6. The results include a description of the performed activities of the method and the results of the activities. Based on the case study, chapter 7 provides with a discussion, conclusion, and future work.

The architect faces the difficult task in choosing logic to locate at server side and logic to locate at client side. The user's device highly influences the user experience. Because the used device is unpredictable, the architect has trouble substantiating the placement of logic. The following sections describe the structured approach taken to provide with the solution to the main problem.

2.1 Research Questions

Based on the prior stated objective and problem statement, we define the main research question as follows:

RQ *How can Software Operation Knowledge assist in predicting the effect of changes in the software's architecture on user-perceived latency of web applications?*

Answering the following sub questions aid in answering the main research question. The following paragraphs describe the subquestions which together provide with an answer to the main research question. We see the first step in answering the main research question in researching the concept of user-perceived latency (UPL) together with the metrics of UPL. We formalize the first research question as follows:

SQ1 *What data is needed from the usage of web applications to obtain Software Operation Knowledge on user-perceived latency?*

The user's device influences UPL. The architect only knows the user's device when the user interacts with the application. After the developers completed the application, it is placed into a production environment where the user can work with the application. Therefore, we use a production environment to obtain values for the metrics of UPL. To obtain these measurements, we instantiate and use the software operation knowledge (SOK) framework. We define the second research question as follows:

SQ2 *In what way can data, used to gain Software Operation Knowledge on user-perceived latency be obtained from the users of a web application?*

After the architect has defined the metrics and values have been obtained, they need to analyze these measurements. We use data mining techniques to find structures and patterns in these measurements. The architect can use process mining techniques on these patterns and structures to create information about UPL. This leads to the third research question:

SQ3 *How can data mining and process mining be used on the data to obtain Software Operation Knowledge on user-perceived latency?*

After data mining has found structures and patterns and process mining has provided with information about UPL, we research how the architect can use the information to improve UPL. We define the fourth research question as follows:

SQ4 *How can Software Operation Knowledge improve user-perceived latency?*

Answering the first four sub questions, result in a number of tools. We can put these tools into practice to define UPL metrics, obtain values for the metrics, analyze the metrics, and identify improvements in UPL. Based on the identified improvements, the architect changes the architecture. Before the architect puts the application into production, they need to measure the effect of the architectural changes in a test environment. We need to research what data we can use to create a test environment. We formulate the fifth research question as follows:

SQ5 *What data needs to be obtained from the users of a web application to create a test environment which simulates a measured user-perceived latency situation?*

After we defined the data needed to create a test environment, we need to define how we can obtain this data. We define the sixth research question as:

SQ6 *How can a structured approach describe the creation of a test environment and the obtaining of the necessary data?*

After we defined the necessary data to create a test environment and described how the architect can obtain data, we can make a testing environment. To predict the effect of changes in the architecture on UPL, we first need to recreate UPL on the test environment. When we obtain similar or comparable measurements in the test environment as in the production environment, we consider UPL recreated. We define the sixth research question as follows:

SQ7 *Which actions need to be performed to recreate user-perceived latency on a testing environment?*

After we provided with steps to define data, obtain data, and measure data to set up a testing environment, we need to predict the effect of changes in the architecture and software on UPL. We define the eighth research question as follows:

SQ8 *How can the recreated user-perceived latency on a testing environment be used to give a prediction on the actual improvements of user-perceived latency of web applications?*

Answering questions 1 to 4, result in tools to define, measure, analyze, and improve UPL. Answering questions 5 to 8 result in tools to define, measure, recreate, and predict UPL in a test environment. We combine the tools and steps to use these tools, to create a method. The method provides the architect with structured approach to measure UPL, improve UPL, and predict the result of architectural changes on UPL. We define the last research question as follows:

SQ9 *How can a method be created to describe the process from Software Operation Knowledge on user-perceived latency to using Software Operation Knowledge for improving user-perceived latency to predict the effect of architectural changes in the software?*

After we answered the research questions, we defined the extend to which SOK can assist in predicting the effect of changes in the software's architecture on user-perceived latency of web applications. The first 4 research question provide information on how an architect can use SOK to obtain improvements in UPL. These improvements lead to architectural changes in the web application. The sub questions 4 to 8 describe how an architect can create a testing environment which assist in predicting the effect of architectural changes on UPL. The last research question provides with a structured approach to apply the tools described in questions 1 to 8. We define this structured approach as a method.

2.2 Research Method

To answer the stated research questions, we use the seven guideline as described by Hevner, March, Park, and Ram [10]. Hevner et al. characterize two paradigms of research in the Information Systems discipline. The behavioral-science paradigm is described as a research method which focuses on searching what is true. The other paradigm, design-science, focuses on finding what is effective. Because we

validate the method by creating a tool to gain knowledge about UPL, we categorize the research method of this research as a design-science approach. Figure 2.1 presents the Information Science Research Framework presented by Hevner et al. [10]. We applied the figure to adhere to the current situation of the research project. The framework describes how the foundations and methodologies of the knowledge base, the organizations, and technology of the environment serve as a base to develop theory, an artifact, and the justification and evaluation criteria of the information systems research. As an addition to the framework, Hevner et al. [10] describe seven guidelines. The following paragraphs describe these guidelines together with their relationship with the research.

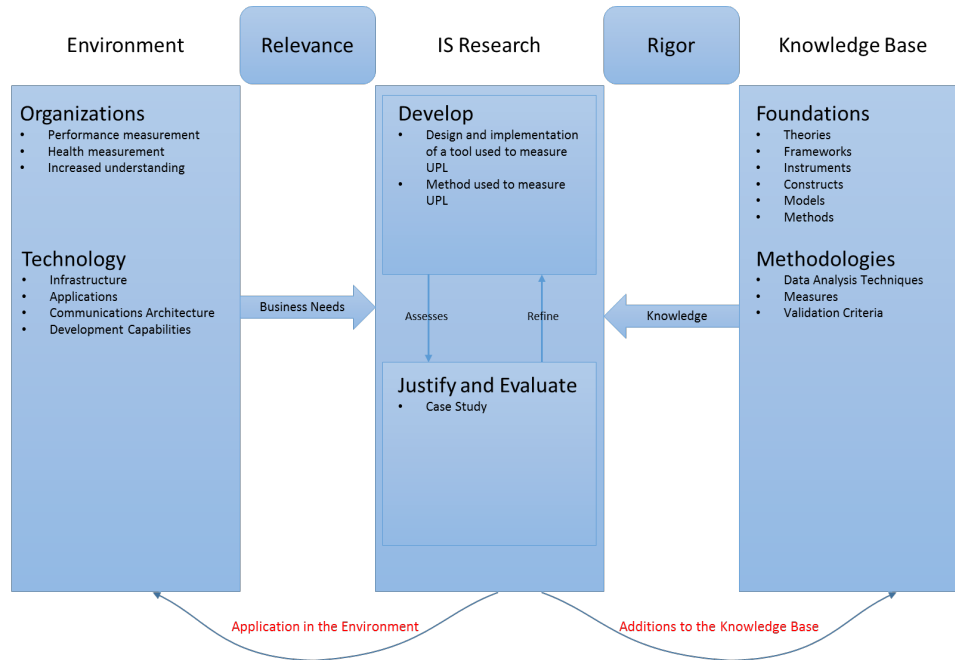


Figure 2.1: Information Science Research Framework [10]

Design as an Artifact The guideline prescribes how the product of a design-science research must be a viable artifact. An artifact can be a construct, model, method, or an instantiation.

- The first artifact created is a design and implementation of a tool which the architect uses to measure UPL of a web application.
- The second artifact describes a method on how the architect can use the tool to gain knowledge about UPL, to create a test environment, and establish a prediction on the effect of architectural changes in the software on UPL.

Problem Relevance Development of technology-based solutions is the objective of design-science research. The research is based on important and relevant business problems. AFAS Software B.V. has triggered this research project. The company is rebuilding one of their applications. The main problem experienced during this rebuild is the lack of an accurate prediction whether the rebuild is an improvement of the prior application or not. In the previous chapter, we described additional indications that the research is based on important and relevant business problems. The method aids the architect when implementing the tool used to gain knowledge about UPL. The test environment allows for predicting the effect of changes in the software’s architecture. Both the tool and the method are technology based solutions.

Design Evaluation The design evaluation guideline prescribes that the design artifact’s utility, quality, and efficiency should be demonstrated through well-executed evaluation methods. The first research object is to define the concept of UPL and modifying the SOK framework so that we can apply the framework to UPL. Multiple expert reviews validate the result of the first step. We create the method

based on the modified SOK framework. Through a case study, the method is validated. During the case study we use this method to measure and improve UPL. We conducted the case study at AFAS Software B.V. where the architects of the company guide the research. The experts at AFAS B.V. validate the case study.

Research Contributions To be effective, the design-science research must provide verifiable contributions. These contributions must be in the areas of the design artifact, design foundations, and/or design methodologies. In this research we produce research contributions encompassing a method based on the SOK framework. An architect can put the method into practice to create a test environment alike a production environment and predict the real world effects of the software’s architectural changes on UPL.

Research Rigor Research rigor concerns the use of rigorous methods in both the construction and evaluation of the design artifact. We perform a literature study to obtain the metrics measured by the tool. Expert reviews evaluate the metrics and a case study further evaluates the correctness of the method.

Design as a Search Process When searching for the most effective design artifact, the artifact must utilize the available means whilst satisfying the legislation in the problem environment. When obtaining data needed to gain information about UPL, the tool gathering the data must be performing as optimal as possible. Because the tool adds another layer in the web application, performance of the application might decrease. Therefore, the tool has to be efficient. The case study points out whether the tool is efficient enough.

Communication of Research The researcher must communicate the design-science research to technology oriented and management oriented audiences. We choose a presentation as the medium of communication. We perform a presentation to a technology oriented audience and a presentation to management oriented audiences, after completion of the method, at AFAS Software B.V. Both presentations consist out of two parts, theory and practice. The theory section elaborates on the ideas, and concepts used in the tool, method and the process on how the method was created. The practice section gives a demonstration of the tool.

2.3 Challenges

Web applications can widely differ in the containing business logic. Metrics used in one application might not be applicable in another application. Therefore we see a challenge in creating tools to help the architect define metrics applicable to the specific situation. We see another challenge when creating the tool used to measure the specified metrics. We know the implementation has to be efficient for performance reasons. But we do not know the amount of measurements and the number of users. Therefore we do not know the amount of data this implementation needs to process. Creating tools aiding in the implementation of the tool gathering and processing the measurements is a challenge. Using data mining and process mining in combination software architecture is a topic which has not been widely covered in literature. It can be challenging performing data mining and process mining techniques on a data set where these techniques have not yet proven their added value. After the elements that affect UPL are known, stored, and analyzed, we need to research how these stored elements can aid in the creation of a test environment similar to the production environment. After changes have been made in the architecture and software, the possibility emerges that certain processes have been changed. Because the recorded functions cannot be replayed in the same way they have been stored, changes in the mentioned processes make more difficult to recreate UPL in a test environment. Therefore we need to research how to cope with these changes in the functions. The last challenge is to combine the theory and practice. Because of multiple stakeholders, academic and corporate, it is challenging satisfying both.

2.4 Final Deliverable

The research sets out to find the extend to which SOK can assist in predicting the effect of changes in the software’s architecture on UPL of web applications. To find this extend, we investigate SOK and the SOK framework. Based on this research we create a structured approach in the form of a method. The architect can use the method to measure UPL, improve UPL, and predict the effect of architectural changes on UPL.

2.5 Project Planning

We use a Process Deliverable Diagram (PDD) as described by Van der Weerd and Brinkkemper [12] to depict the activities performed and their corresponding deliverables, see Figure 2.3. We created the PDD by using a combination of Meta-Process modeling (an adaption of the UML activity diagram) and Meta-Deliverable modeling (a concept diagram). The PDD notation consists out concepts and activities which can be standard or complex. A complex concept or activity can be either open or closed, see Figure 2.2. The following paragraphs elaborate on the PDD notation.

Standard Concept A standard concept contains no other concepts [12].

Open Complex Concept An open complex concept consist out of multiple other concepts. The PDD depicts the (sub)set of these concepts within the same PDD. The open complex concept thus consists out multiple other known concepts [12].

Closed Complex Concept The closed complex concept consists out of multiple containing concepts which have been deliberately omitted. The difference with the open complex concept is that the containing concepts of a closed complex concept are unknown [12].

Standard Activity A standard activity contains no sub activities [12].

Open Complex Activity An open complex activity consist out of multiple other known activities. The PDD depicts the (sub)set of these activities within the same PDD [12].

Closed Complex Activity As with an open complex activity, the closed complex activity contains multiple other activities. The only difference with the open complex activity being that the containing activities of the closed complex activity are unknown [12] [12].

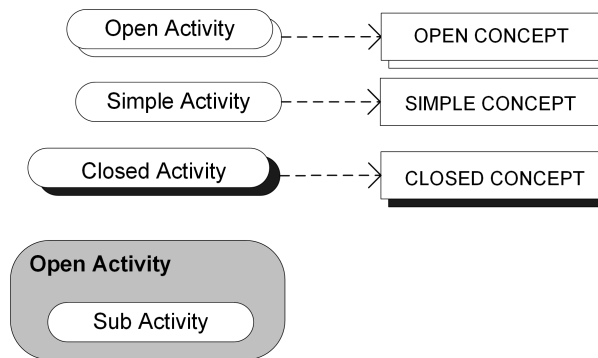


Figure 2.2: A visualization of an open activity and a resulting open concept

We further elaborate on the activities in Appendix A. Appendix B describes the deliverables.

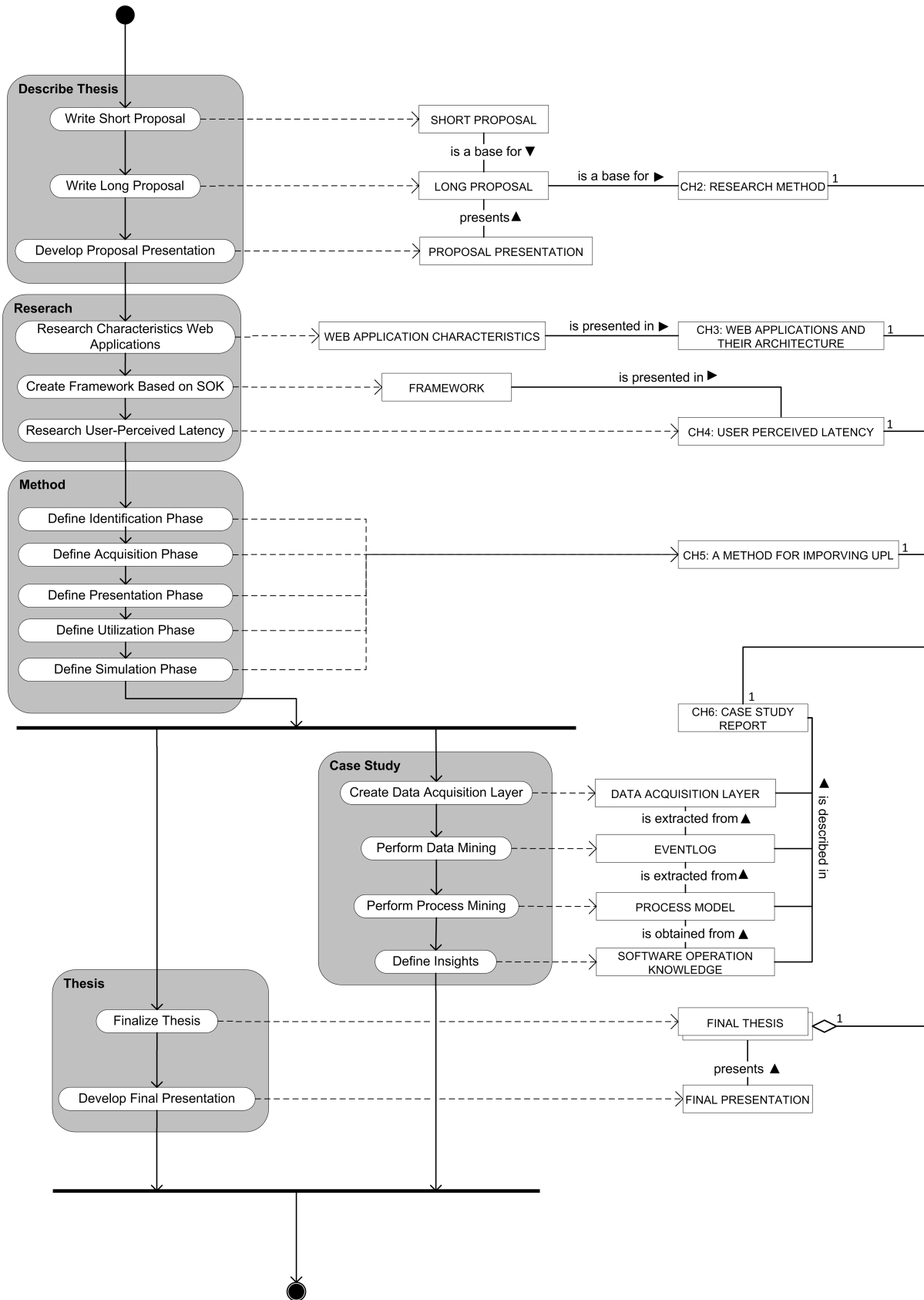


Figure 2.3: Process Deliverable Diagram of this research planning

Web Applications and Their Architecture

Before we can measure user-perceived latency (UPL) at the client, the elements of web applications and the architecture needs describing. The following sections elaborate more on these elements. We describe a running example which we use as an example throughout the rest of this thesis. This chapter describes Asynchronous JavaScript and XML (AJAX) as a means of communication and the relation between AJAX and two web application models. Furthermore, a description of the document object model (DOM), software operation knowledge (SOK) and the SOK framework are stated.

3.1 Running Example

As a running example we consider a web application in the form of a warehouse administration application. The application consist of functionalities used to manage products in a warehouse and the users of the application. The warehouse manager can use the application to see the stock of a product and restock if necessary. Employees can use the application to sell, bill, and ship the products. Administrators can use the application to manage the employees and warehouse managers (i.e. create new and provide them with the appropriate rights, alter existing rights, and delete rights). Figure 3.1 uses an UML use case diagram to depict which functions a type of employee can execute.

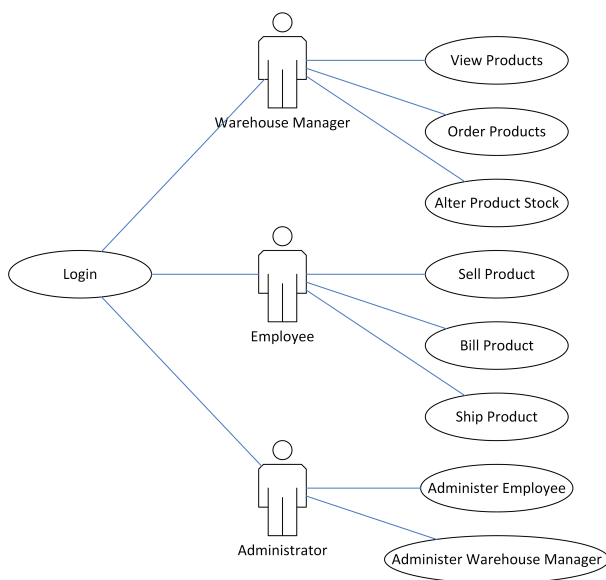


Figure 3.1: Functions of the Administrative Web Application

The application consists of client where the application presents a user with a login. The login function identifies one of the three type of users (i.e. warehouse manager, employee, or administrator).

After identification, the application provides the user with the appropriate page. The user can now use the application to perform the functionalities the user has the rights to perform. On the client, the JavaScript program performs functionalities (e.g. show the appropriate user interface and form validation) and communication with the server is performed. The JavaScript program contains three components (one for each type of user). Figure 3.2 depicts the architecture of the application. The application stores and obtains users and products from a server. For communication with the server, the application uses AJAX. The following section describes this form of communication.

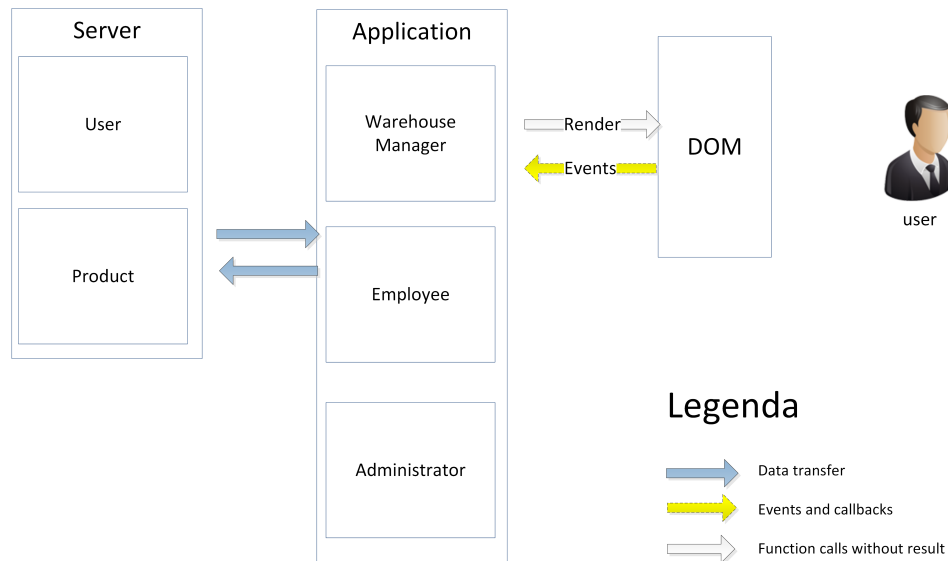


Figure 3.2: Architecture of the Running Example

3.2 AJAX Web Application Model

AJAX, or Asynchronous JavaScript and XML, is a combination of multiple technologies that have created a fundamental shift in the possibilities on the Web [2]. As described by Garret [2] AJAX incorporates:

- *Standards-based presentation using XHTML and CSS;*
- *Dynamic display and interaction using the Document Object Model;*
- *Data interchange and manipulation using XML and XSLT;*
- *Asynchronous data retrieval using XMLHttpRequest;*
- *JavaScript binding everything together.*

Correct usage of these technologies can improve performance and user experience of web applications. With AJAX, developers can create desktop-like applications on the web, which initially was not designed for these kinds of applications [13]. Therefore, web applications are becoming more attractive to organizations. AJAX is also attractive to use for software architects because AJAX can increase user interactivity, improve user-perceived latency, increase scalability of the client, increase adaptability, and support different platforms [14]. In the running example, the application communicates between the client and the server with AJAX.

Around 2005, developers used AJAX in a new web application model [2]. Based on the new web application model, developers use AJAX to focus on asynchronous interaction. The asynchronous interaction allows for an improved user experience and performance by refreshing only a part of the web page. Since 2005 we see a shift from synchronous interaction to asynchronous interaction, like in the AJAX web application model [2]. Figure 3.3 shows the differences between these two types of interaction.

Looking at the synchronous model at the top of the figure, the client directly redirects the user activity to the server where the server responds to the user activity by showing the user a different screen. The asynchronous model at the bottom of the figure has divided the client into two parts, the Browser's UI and the AJAX engine. Users interact with the Browser's UI which directs the interaction to the AJAX engine. A response to the interaction can be given by the AJAX engine in following two ways. The AJAX engine can respond directly the user e.g. expand a menu item.

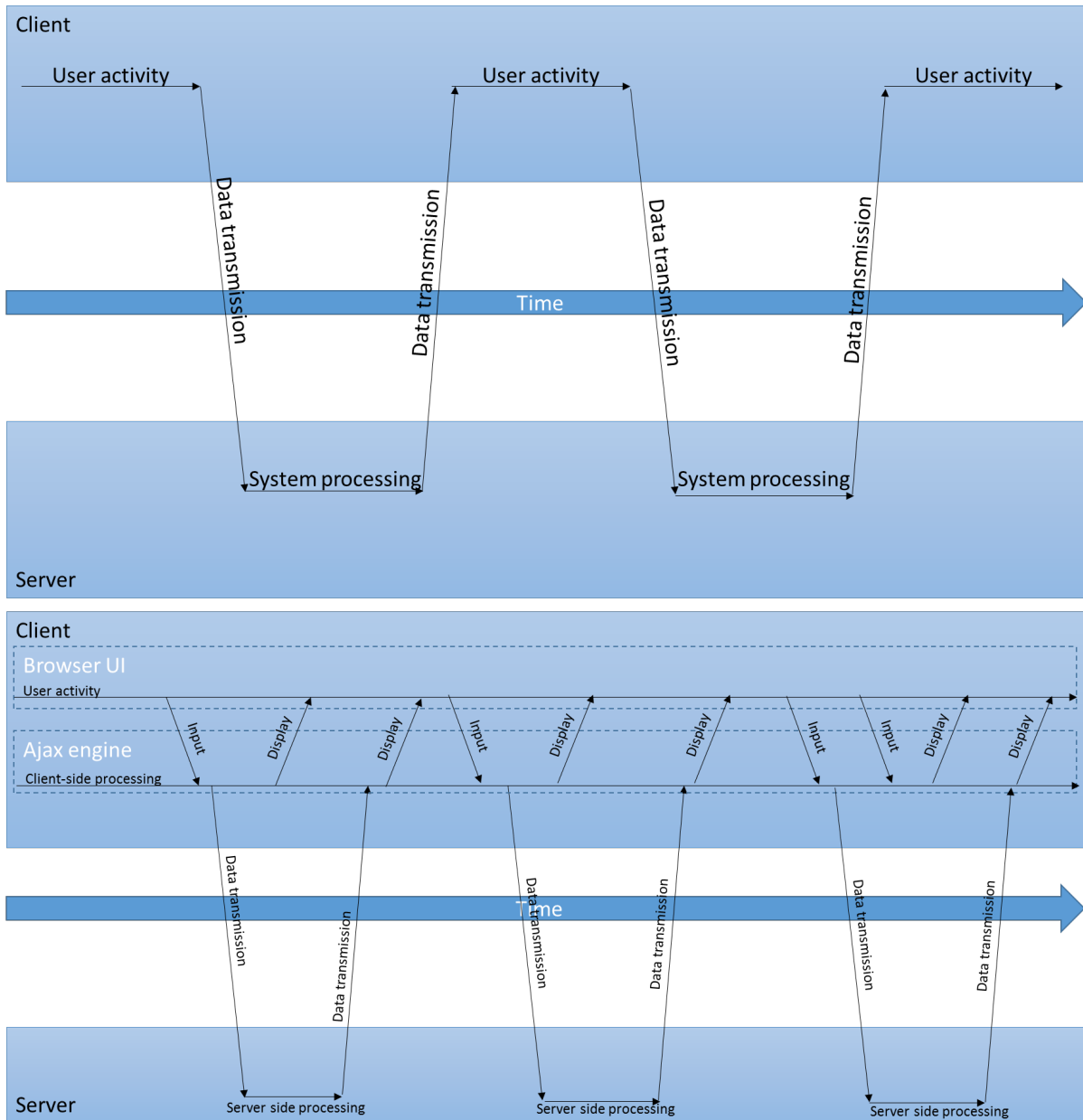


Figure 3.3: The difference between the synchronous model (top) and the asynchronous model (bottom) [2].

The AJAX engine can partially respond to the user, use AJAX to get the rest of the response from the server, and use the server's response to fully respond to the user. As an example, a user requests to search, the AJAX engine partially responds by showing a loading indicator. The AJAX engine then sends an AJAX request to the server, which responds with the search result. The AJAX engine removes the loading indicator and shows the list. A difference being the synchronous model directing the user activity directly to the server, the asynchronous model handles the user activity by a JavaScript program.

The asynchronous model allows for the application to use both the client and server more efficiently. The increased efficiency leads to an increased performance of both the client and the server. To communicate the results of the processed actions back to the user, the JavaScript program uses DOM manipulations which we explain in the following section.

Looking at the running example, we can see the increased interactivity in form validation. When logging into the application, the application checks whether both the user name and password fields have been filled in or left empty. Looking at the synchronous model in Figure 3.3, the application would directly send the fields to the server where the server would check whether the fields have been filled in or not and act accordingly. When the user leaves a field empty, the application shows the login failed screen. Looking at the asynchronous model, the application can handle an empty field more efficiently. The AJAX engine checks whether the user has filled in both field or not. When the user has filled in both fields, the client makes an AJAX call to the server. The asynchronous model is more efficient because the communication between the client and server includes overhead e.g. the client needs to initiate a request, send the request to the server, the server needs to read the message, perform logic, initiate a response, send the response back to the client, the client needs to read the response, and the client needs to show the result to the server. Because of the added overhead, the architect can use the client to respond to user interaction usually faster compared to the server. Note that because of security reasons, the architect cannot always use the client for validation, for instance when the application checks whether the user has filled in a correct username and password. The architect needs to decide where to perform which logic and which application component can perform the logic the most efficient?

Web applications use the Document Object Model (DOM) to add, delete, and manipulate nodes in a HTML document. Web applications usually use JavaScript to perform these operations. Nicol, Wood, Champion, and Byrne [15] define the DOM as:

The Document Object Model (DOM) is an application programming interface API for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

Figure 3.4 shows a partial example of a valid HTML document. A graphical representation of the partial HTML document shows the tree structure of nodes in Figure 3.5.

In conclusion, Garret [2] describes AJAX as a combination of technologies which web applications use in combination with a JavaScript program to communicate from the client to the server and back. The AJAX technology makes it possible to create more interactive websites, which one can see in the difference between the synchronous and asynchronous model. A key component of the increased activity is the use of DOM manipulations by the JavaScript program.

```

<table>
  <th>
    <td>
      Name
    </td>
    <td>
      Date of Birth
    </td>
  </th>
  <tr>
    <td>
      Rick Hoving
    </td>
    <td>
      22-10-1990
    </td>
  </tr>
  <tr>
    <td>
      John Doe
    </td>
    <td>
      01-10-1976
    </td>
  </tr>
</table>

```

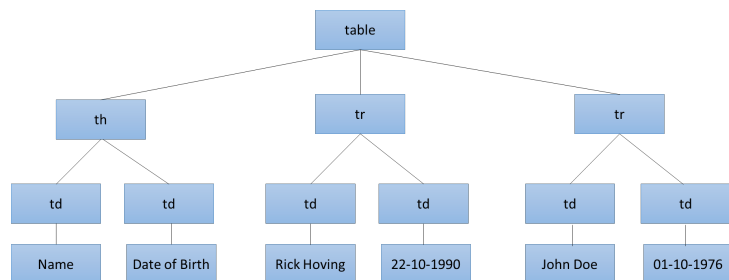


Figure 3.5: Graphical Representation DOM Document

Figure 3.4: Example HTML Table

3.3 Software Operation Knowledge

When creating and implementing an architecture, the architect faces the difficult discussion on where to locate which logic. We see the uncertainty in the user experience of the application as the biggest problem faced by the architect in making this decision. To gain knowledge about the result of the architect's decision on whether to locate the logic on the client or the server, the architect needs to be provided with information about the application's behavior. To provide the architect with this information, we use the Software Operation Knowledge (SOK) framework. This section gives a description of SOK, the SOK framework, and how the framework aids in providing information about the user experience. Van der Schuur [16] states that gaining SOK is central in the SOK framework. He describes SOK as a type of knowledge used to gain information about the behavior of an application at run time. One can use SOK for a variety of purposes e.g. bug prioritization, see if an update has actually fixed a bug, or application usage. Van der Schuur, Jansen, and Brinkkemper define SOK as [16]:

Knowledge of in-the-field performance, quality and usage of software, and knowledge of in-the-field end-user software experience feedback.

The SOK life cycle, depicted in Figure 3.6, consists of 5 processes: identification, acquisition, integration, presentation and utilization [16]. During the *identification process*, the SOK utilization goal, and operation knowledge demands associated with the SOK utilization demands are identified. The *acquisition process* consists of multiple steps. First, one makes a translation from the end-user behavior to software operation data. During the next step, one transfers the software operation data to the software vendor. The last step of the process concerns the identification and extraction of software operation data from software operation data sources. During the optional process (the *integration process*) one integrates the resulting software operation information into the existing processes and infrastructures of the software vendor, like plug-ins, conversion components or mediator services. The software operation information concerns the information resulting from the *acquisition process*. During the *presentation process* one presents the software operation information. The architect uses graphs, diagrams or other presentation artifacts depending on suitability for a particular framework perspective to perform the presentations. The last process, the *utilization process*, describes processes and response actions to effectively gain SOK [16].



Figure 3.6: The SOK Life Cycle [3]

Looking at the running example, we can define SOK as e.g. the errors and type of errors given in a certain timespan, the frequency of function execution, the devices (computer, tablet, or mobile) used to initiate a function. Knowledge about the errors and type of errors can lead to a prioritization of bugs and see if a bug has actually been fixed e.g. when the error does not occur anymore, the bug has been fixed. We find knowledge about frequency of function execution useful when prioritizing optimizations for the function which was used the most. The architect can use knowledge about the device initiating a function to optimize the function for a specific device e.g. when the user mostly interacts with the application on a tablet device, the architect can optimize the application for tablets.

Figure 3.7, describes the processes of the SOK life cycle. In the figure, software operation data flow, the information, and knowledge through software vendor tools and processes are depicted [16]. The SOK framework distinguishes two type of stakeholders, the software vendors and the customers. The framework considers both business-to-consumer customers and business-to-business customers of the software vendor.

The SOK framework incorporates three perspectives. These perspectives are: the development perspective; company perspective; and customer perspective [3]. The development perspective contains processes concerning the production of software products. These products concern products which the architect can deploy at customers. The company perspective focuses on processes which have no direct relation to the development, such as sales. The customer perspective concerns processes which influence existing relationships between customers and software vendors, such as support.

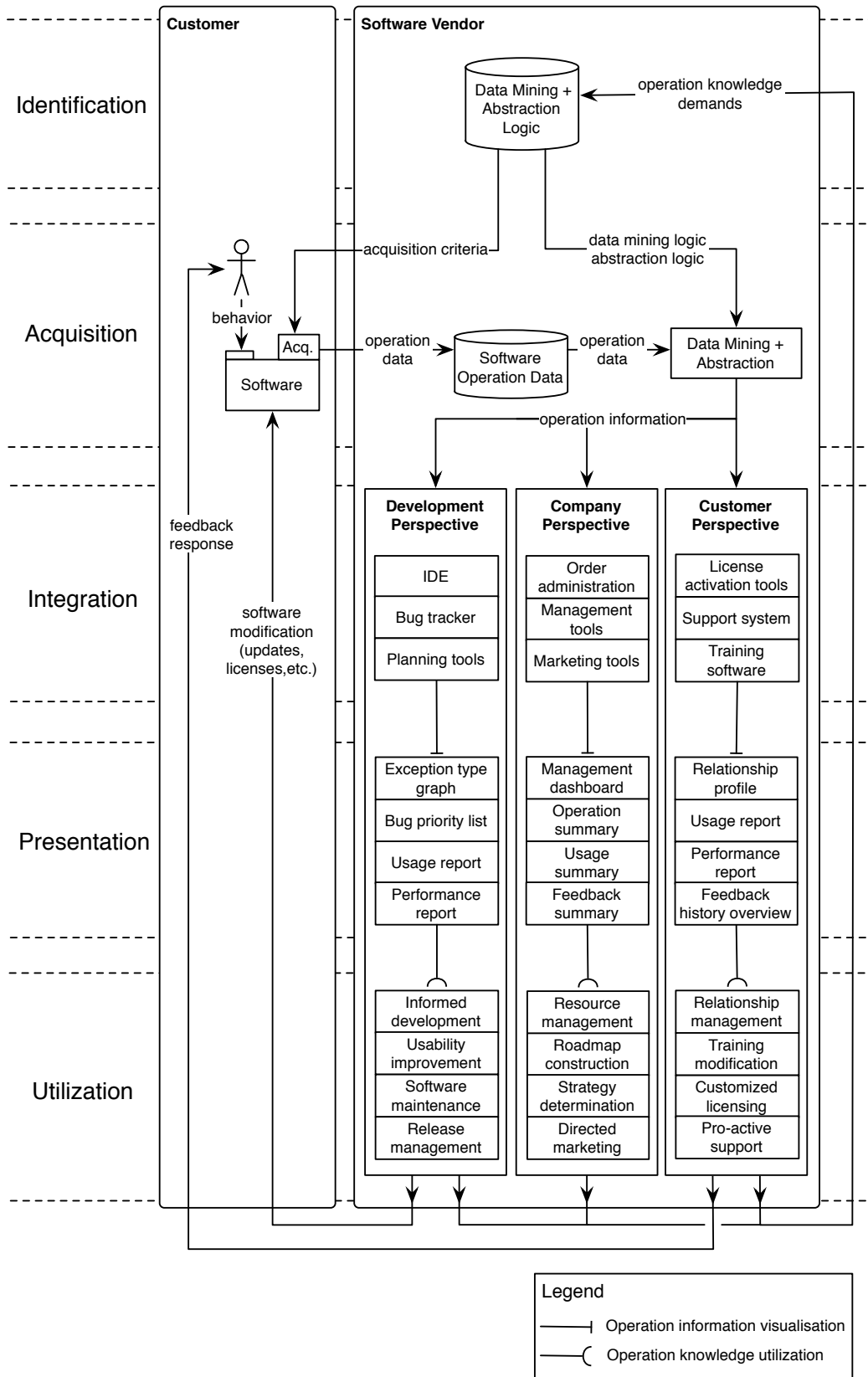


Figure 3.7: Software Operation Knowledge Framework [3]

User-Perceived Latency

A proper placement of logic (client side or server side) can influence the user experience of a web application. Having a proper balance between logic on the client and on the server increases the user experience. This chapter provides with more information about a metric which is highly influenced by the placement of logic, namely user-perceived latency (UPL). First we use the four stages of human computer interaction to define the concept of UPL. We use these stage to divide UPL in, Human Interaction Time (HIT) and System Response Time (SRT). After we have described UPL, the rest of this chapter focuses on SRT. We focus on SRT because of the architectural perspective of this research. Next, we describe the relation between the SRT metric and multiple quality attributes. To define metrics which we can use to measure SRT, we perform a literature study on SRT. Lastly, the chapter describes the software operation knowledge (SOK) framework and how the architect can use the framework to measure SRT.

4.1 Human Computer Interaction

The area of Human Computer Interaction (HCI) is widely covered by research. Newel and Card [17] divide HCI into the following disciplines: computer graphics, human factors, cognitive psychology, and artificial intelligence. The following paragraphs will describe research papers within each of these disciplines.

In the area of computer graphics Hartson and Gray [18] describe the User Action Notation. The architect can use the User Action Notation to create interface designs based on the user interaction instead of the functionalities of the software [18]. The notation allows for the modeling of both synchronous and asynchronous interaction. Modeling based on the functionalities of the software only allows for synchronous interaction design [18]. Maulsby, Witten, Kittlitz, and Franceschin [19] propose the use of a system which allows for the user to create additional functionalities by providing the application with examples. The user has the ability to learn the system, named Metamouse, new functionalities by interacting with the system.

In the area of human factors, Riedl [20] has found a correlation between a stressful application and the stress hormone cortisol, concluding that applications can induce stress to a human. The same effect was also found in another paper by Riedl, Kindermann, Auinger, and Javor [21].

In the area of cognitive psychology, Kortum and Peres [22] have researched the relationship between the perceived usability and the effectiveness of the system. Newel and Card [17] and Carrol [23] describe the role of psychology in HCI.

In the area of artificial intelligence, Fischer defines techniques for systems to create user models. Fisher describes these techniques for high functionality applications, knowledge based systems, design environments, and critiquing systems [24].

Looking at the disciplines of HCI, and papers written in the area of HCI we see a focus on measuring, and improving the human's experience with an application from the human's perspective. We see another focus on measuring and improving the human's experience from an interface perspective. Lastly, we see a focus on improving the human's experience with the application from a functionality perspective. Although research has widely covered HCI, and has researched HCI from multiple perspectives, we

identify a lack of research performed from an architectural perspective.

To provide the architect with information about the user experience, we specify a set of activities a human performs when interacting with computers. Using these steps, we identify the steps at which the software's architecture influences UPL. Providing with information in these steps from an architectural perspective aids the architect in improving the software's architecture. The improvement results in an improved experience of the human with the application.

In human interaction with computer applications, Norman [25] has identified a cycle of four stages. In these stages, intention is defined as *'The internal, mental characterization of the desired goal'* [25, 26]. Selection is defined as *"The stage of translating the intention into one of the functions possible at the moment."* [25]. These functions concern a (set) of command(s) which the application understands. The executing of the function concerns the entering of the selection of command(s) into the application. After the application has completed these command(s), the outcome is projected and is evaluated by the user. Evaluating of the outcome allows the user to form the intention to selecting and executing another function [25]. Dix, Finlay, Abowd, and Beadle [26] describe HCI using the following set of 7 activities to divide both evaluation and execution.

- *Establishing the goal*
- *Forming the intention (more specific than goal)*
- *Specifying the action sequence (based on intention)*
- *Executing the action*
- *Perceiving the system state*
- *Interpreting the system state*
- *Evaluating the system state with respect to the goals and intentions* [26]

Using the steps defined by Norman [25] and Dix, Finlay, Abowd, and Beadle [26] we define the following steps in which a human interacts with a computer.

Forming the intention In this step, the human defines a goal and characterizes the functions e.g. create user, go to page, search person, and save changes.

Selecting a function Based on these functions, the human will make a selection in this step.

Executing the function In this step, the human executes the selected functions. When a human interacts with a web application, the used device provides multiple means of selecting and executing a function. For instance, when the human interacts with the application using a desktop, the human can use the mouse or touch-pad to select a function and click to execute the function. On a phone, the human can use the touchscreen to select and execute by tapping the touchscreen.

Evaluating the outcome The result from the previous step, the human has put the system in another state. In this step, the human perceives the new state, interprets the new state, and evaluates the new state. After the human has evaluated the state of the system, the human sets a new goal to achieve.

Figure 4.1 depicts these steps using an UML activity diagram. Looking at the running example, during the first step an employee sets a goal to ship a product. Next, the employee looks at the functionalities offered by the application and selects the ones which the employee can use to ship a product. These functions are ship product by single product order and ship product directly. In the second step, from the categorization, the employee selects the ship product directly function by moving the mouse to the button which will execute the selected function. During the third step, uses a left mouse-click to execute the function. The system responds by changing the state. The new state presents the employee with the text "shipment send". During the fourth step, the employee perceives the system has sent a message, interprets the text "shipment send" as no errors have occurred, and evaluates that the shipment has been sent. The employee then proceeds to form the intention to perform another function in the first step. Note that because we consider this as a never ending continuous process, the Figure has no end state.

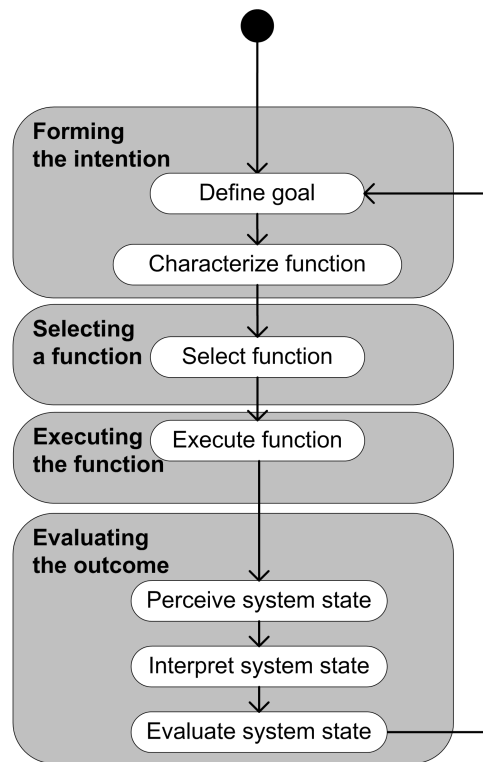


Figure 4.1: The Four Stages of Human Computer Interaction [25, 26]

4.2 Characteristics of User-Perceived Latency

We describe UPL as the user’s perception in the application’s delay. Looking at, and using the terminology of the prior mentioned steps in human computer interaction, we define user-perceived latency as:

”The time between the start of selecting a function and the end of evaluating the response from the application by the user”.

Looking at the definition of UPL, we define the following factors influencing UPL: the person interacting with the application, the experience of the user with the application, the execution time of the application, and the user interface of the application [25]. Figure 4.2 gives a graphical explanation of UPL using an UML timing diagram. In the figure, the horizontal line (the lifeline) indicates the state of the actor (active or inactive). Looking at the lifeline, during the first two stages of human computer interaction, the user is active. The arrows represent communication between the actors e.g. the user clicks on a button, or the system presents the user with a message. Note that in the figure, the actor’s lifeline can switch between active and inactive e.g. a keep alive call of the system, or the user being distracted by an add. For the readability of the figure, we have omitted this from the figure.

As depicted in Figure 4.2, we divide UPL into two parts, a part where we see the human active and a part where we see the system active. We name these two parts, the Human Interaction Time (HIT) and the System Response Time (SRT). In other words, we define HIT as UPL from a user’s perspective and SRT as UPL from an architectural perspective. We define HIT as:

”The time the user needs to select and execute a function, and the time the user needs to evaluate the outcome of the system’s response”.

A definition of SRT encompasses the time the application takes from the user’s execution of a function to communication of the function’s result to the user. Mesbah and Van Deursen [14] describe SRT as:

”The period between the moment a user issues a request and the first indication of a response from the application”

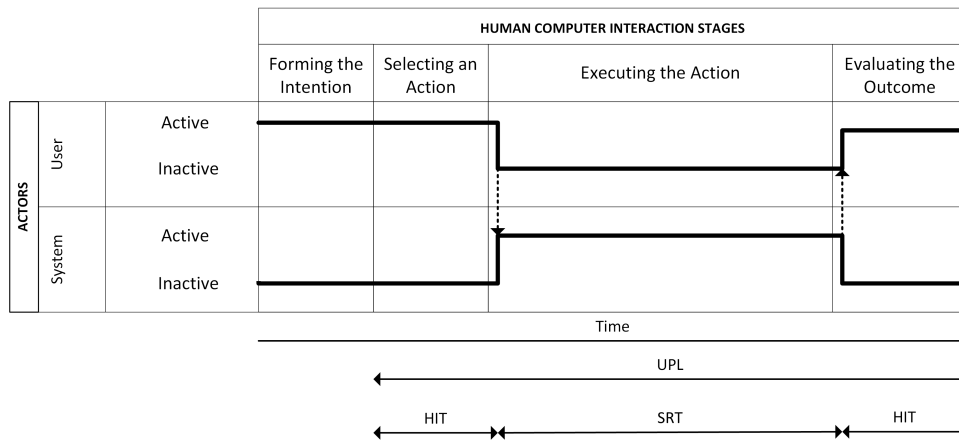


Figure 4.2: A Visualization of the Human Computer Interaction Process

Although this definition encompasses an essence of SRT, we disagree with its scope. Take the following example, a user clicks on a button, the application shows a loading indicator, and the application shows the result on the screen. Looking at the prior stated definition of UPL, one can measure the duration of UPL as the time between the click of the user and the showing of the loading indicator. However, the user has experienced a latency until the application shows the entire result on the screen, not the loading indicator. The research therefore builds on the definition of Mesbah and Deursen and defines SRT as:

”The period between the moment a user executes a function and the last response from the application”.

In the running example, we can see HIT in the duration of the employee pointing the mouse to a button of the send shipment function, clicking the mouse to execute the function by pointing, reading the system’s response and evaluating that the shipment has been sent. We can see SRT in the execution of the send shipment function and presenting the employee with the text ”shipment send”.

In conclusion, the architecture does not influence HIT. The user’s subjectivity and experience influences HIT. However, the architecture does influence SRT, the architect can use the architecture used to measure SRT, and the user’s subjectivity and experience do not influence SRT. Because of the architectural focus of the research, we focus on SRT. Measuring SRT provides the architect with information on the quality of the system.

4.3 System Response Time and Quality Attributes

In the previous section, we conceptualized UPL and have defined HIT and SRT as two components of UPL. Because of the architectural focus of this research we focus on improving the system’s quality by improving SRT. To measure the quality of software applications, quality attributes are used [27]. Bass, Clements, and Kazman describe a quality attribute as:

A measurable or testable property of an application that is used to indicate how well the application satisfies the needs of its stakeholders [27]

Looking at the definition of Bass et al. [27], because SRT is not a property of the application, we do not consider SRT as a quality attribute. However, we do consider SRT as a metric of multiple other quality attributes, see Figure 4.3. The software product quality model (see Figure 4.3) as described by the ISO 25010 describes a categorization of quality attributes into eight categories. The ISO 25010 standard further divides each of these categories into sub-categories [11]. The following paragraphs describe the relation between the SRT metric and the quality attributes described by the ISO 25010 standard.

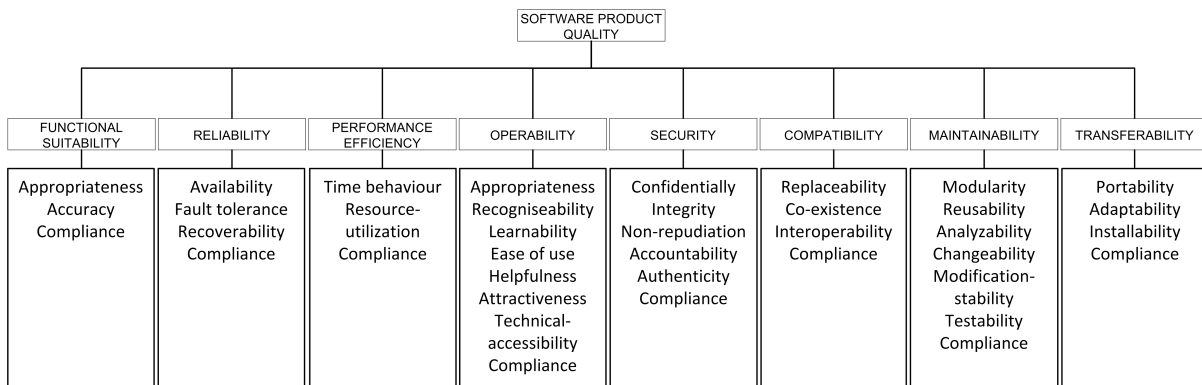


Figure 4.3: The software product quality model [11]

- The ISO 25010 standard describes Functional Suitability as *"The degree to which the software product provides functions that meet stated and implied needs when the software is used under specified conditions"* [11]. SRT can aid in measuring the conditions of the system and in defining the conditions the system has to be in. As an example, the architect measures 200 milliseconds as SRT of a function when they put the system in a high pressure situation. They can use the measured 200 milliseconds to define the condition of the system e.g. we see the system in a high pressure condition when we measure a SRT of 200 milliseconds. Measuring the same 200 milliseconds aids in concluding that the system is in a high pressure state.
- ISO 25010 standard describes Reliability as *"The degree to which the software product can maintain a specified level of performance when used under specified conditions"* [11]. The degree to which a software product can maintain a specified level of performance influences the fluctuations in SRT. When a software product has a high reliability, the measured fluctuations in SRT are smaller than with software products with a low reliability. If, for instance, we measure a function 1.000 times and the measured durations range from 10 milliseconds to 1000 milliseconds, we conclude the system has a low reliability. When the same 1.000 measurements range from 100 milliseconds to 110 milliseconds, we see the system as more reliable.
- ISO 25010 standard describes Performance Efficiency as *"The degree to which the software product provides appropriate performance, relative to the amount of resources used, under stated conditions"* [11]. We translate the appropriate performance to the appropriate duration of SRT. An increased amount of used resources results in an increased duration of SRT. Therefore a high duration of SRT can indicate a high amount of used resources.
- ISO 25010 standard describes Operability as *"The degree to which the software product can be understood, learned, used and attractive to the user, when used under specified conditions"* [11]. A lower SRT within a software product increases the attractiveness for the user to use. When the user finds the software product more attractive to use, the user will have more patience when understanding and learning the software product. For instance, the user will have more patience to learn to use the application when the system responds within 100 milliseconds to the user, than when the application responds after 1.000 milliseconds.
- ISO 25010 standard describes Security as *"The protection of application items from accidental or malicious access, use, modification, destruction, or disclosure"* [11]. When the architect incorporates more security into a function, the measured SRT duration will increase. Take logging into an application as an example. With a low security, the application checks the username and password combination in a database. If the system finds a match, the system verifies the login. In a high security application, when the user logs into the application the client encrypts the username and password. The applications then initializes a secured connection between the client and server. Next, the client sends the encrypted username and password to the server which decrypts the combination. The server then, checks the combination in a database and verifies the login. Therefore a higher security introduces additional steps to perform for a function which increases the measured SRT.

- ISO 25010 standard describes Capability as *"The ability of two or more software components to exchange information and/or to perform their required functions while sharing the same hardware or software environment"* [11]. When the architect measures SRT for a function, having a fast communication between components can decrease the measured SRT. For instance when the components communicate with XML, the architect measures SRT as 50 milliseconds. When these components use JSON when communicating, SRT improves to 30 milliseconds. Therefore, when the components are highly compatible with each other, the architect can measure a faster communication, which decreases the measured SRT.
- ISO 25010 standard describes Maintainability as *"The degree to which the software product can be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications"* [11]. After information is provided about SRT, the architect can improve SRT. When the architect builds the system with a high maintainability, the architect can implement improvements faster. A high maintainability can therefore improve the rate in which the architect can improve SRT.
- ISO 25010 standard describes Transferability as *"The degree to which the software product can be transferred from one environment to another"* [11]. The architect can use SRT as a metric to decide whether the architect can consider application transferable or not. As an example, the architect measures the duration of SRT as 2.000 milliseconds in an environment. When the architect considers 2.000 milliseconds not acceptable in the environment, the architect can consider the software nontransferable to the environment.

In conclusion, we see SRT as a metric. The SRT metric can be used when measuring the quality attributes described in the ISO 25010 standard. Metrics described in the ISO 25010 standard measure the quality of a system. In functional suitability, SRT can be used to measure and define the condition of the system. Fluctuations in SRT aid in measuring the reliability of the system. High fluctuations in SRT can indicate a low reliability and low fluctuations in SRT can indicate a high reliability. Measuring a high duration of SRT can indicate a high amount of used resources. The amount of used resources can be used whilst measuring performance efficiency. Improving SRT can make the application more attractive to use and to learn. When the user finds the application more attractive, operability improves. Adding security affects SRT. When the architect implements additional security functionalities, SRT measurements can indicate the effect of these additions on the system. Improving the capability of the system improves SRT. When the system has a high maintainability, the architect can implement improvements in the architecture, improving SRT, faster. By measuring SRT, the architect can use the measurements to define the transferability of the system. For instance, when a SRT duration exceeds 2.000 milliseconds, the architect can consider the application not transferable to the environment.

4.4 Providing Information About the System Response Time

As we have described, the architect can use SRT as a metric to aid in measuring quality attributes. In this section we describe the metrics which the architect can use to measure SRT. We define the metrics using a literature review. We structure the literature review according to the approach described by Webster and Watson [28]. Table 4.1 shows the concept matrix created as a result of the literature review. The table shows the metrics described in the selected papers. When a paper mentions a metric, the table indicates this with a "√". The following paragraphs describe the metrics using the papers discussing the metrics.

Browser The browser can influence SRT because of differences in rendering engines and JavaScript engines. One rendering engine might be more efficient in rendering DOM manipulations than another. One JavaScript engine might have a more efficient garbage collector, which will decrease the memory usage. The architect can use the memory usage to identify memory leaks, which can lead to improvements in performance, which will lead to a lower SRT duration.

Garbage Collection The garbage collection of the JavaScript engine reclaims memory occupied by strings, objects, arrays, and functions that the application does not longer use [33]. Because the garbage collection is a part of the browser's JavaScript engine, we incorporate the metric in the browser metric.

Article	Metric					
	Browser	Garbage Collection	Processing	Rendering	Initial Application Load Time	External communication
Performance Metrics [29]					✓	
Automated construction of JavaScript benchmarks [30]	✓	✓		✓		
Performance Engineering of the World Wide Web: Application to Dimensioning and Cache Design [31]						✓
Benchmarking Modern Web Browsers [32]	✓		✓	✓		✓

Table 4.1: Concept Matrix with Web Application Metrics

Processing With JavaScript, the architect can develop a lot of functionality to process data. The architect defines which functionality to program, and to implement. Therefore we consider it impossible to redefine which functionalities encompass processing.

Rendering The application needs to communicate the results of client’s functions back to the client. The JavaScript program of web applications use DOM manipulations to show the user results of its functions. The rendering process encompasses the time (in ms) the application takes to perform the DOM manipulations needed to show the result of the user’s functions (in number of DOM manipulations). Because we see rendering as a part of SRT, when rendering improves, the duration of SRT decreases.

Initial Application Load Time When a user visits an URL of the web application, the logic processed on the client is transported from the webserver to the client. After the client has loaded the necessary files, the browser instantiates the JavaScript application and renders the client. Because we consider initial application load time a measurement of SRT, we do not consider the initial application load time as a metric.

External Communication Some logic cannot be performed on a client, like database manipulations and login validations. Some logic has to be processed by the server. Setting up the communication, the communication itself, and receiving the communication takes time. Therefore we consider external communication as a part of the SRT. The external communication is the time (in ms) it takes from the set-up of the communication to the end of the receiving of the communication.

We used a literature study to define metrics used in web applications. We describe the browser, garbage collection, processing, rendering, initial application load time, and external communication as metrics. The differences in rendering engine and garbage collector of the browser influence SRT. The processing functionalities as programmed by the developer influence SRT. We consider initial application load time as a specific measurement of SRT, not a metric. The application performs logic on both the client and the server. By measuring the external communication between the client and server, we can divide a SRT measurement duration in the client and in the server.

4.5 Analyzing System Response Time

In the previous section we have defined metrics which can aid in providing information about SRT. To provide with more information about SRT the architect needs to link the measurements of the described metrics to a specific measurement of SRT. We call a specific measurement of SRT a function. To provide with more information about the improvements of the function, we divide functions into stages. The following paragraphs elaborate more on functions and stages.

Functions We call a measurement of SRT a function. We define user interaction as the starting point of a function. Apart from the first interaction, a function does not include any other user interaction. We define the duration of a function as the time it takes for the system to react to the user interaction.

Stages Dividing a function into stages gives an even more in depth view of the function and improves locating improvements in the function and thus SRT. To provide even more detail, the architect can further divide these stages into even smaller stages and so on until they have obtained the ideal point of detail. Because of the large amount of different applications and logic residing within these applications, the ideal point of detail and type of stages differs for each application.

4.6 Instantiated SOK Framework Fit For SRT

Because of the uncertainties and unpredictabilities that come with SRT, the architect needs to validate improvements before they incorporate these within the application. To make the SOK framework fit for measuring and improving SRT, the research creates an instantiation of the SOK framework, see Figure 4.4. Because of the architectural perspective of this research, the Company Perspective and Customer perspective do not have the focus. In the integration phase one integrates the resulting software operation information into the existing processes and infrastructures of the software vendor, like plug-ins, conversion components or mediator services. Because existing processes and infrastructures used to measure, analyze, and improve SRT do not exist yet, we have omitted this phase. The following paragraph describes the new components of the framework.

After the architect has identified improvements in the architecture and has altered the application, they need to test whether a change in the application's architecture actually improves SRT. Therefore we added the *simulation process* to the framework. To test the effect of these architectural changes, in the *simulation process* we simulate SRT. To simulate SRT, the simulation process uses the data obtained during the *acquisition process*. The application gathers and processes the data produced by the simulated user using the same process as described by the SOK framework. The simulation process adds a sub-cycle to the framework, the testing cycle. In the testing cycle, the architect can test the effect of changes in the architecture (through the simulation layer). When we obtain the expected effect of the changes, the architect can distribute the application to the customer.

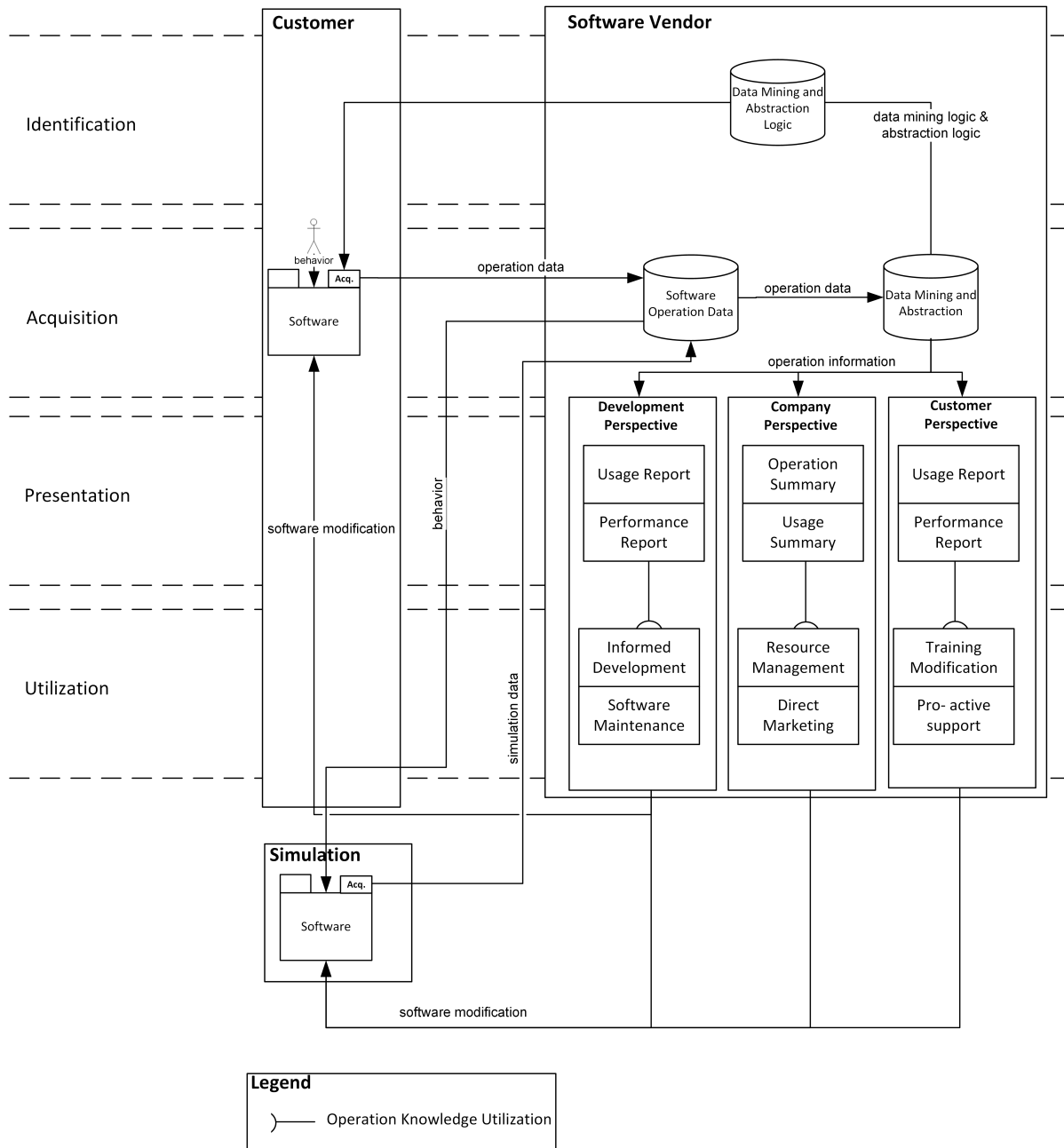


Figure 4.4: Software Operation Knowledge Framework Instantiated for System Response Time Measures

A Method for Improvement of User-Perceived Latency

To improve user-perceived latency (UPL), in this chapter we present a structured approach in the form of a template method, see Figure 5.1. The architect can use the method to measure, predict, and improving system response time (SRT) in web applications. The following sections describe how the architect can use the method and provide with tools they can use whilst using the method. We based the method on the instantiated software operation knowledge (SOK) framework fit for SRT measures. The method contains five main activities, four of which related to the four processes of the SOK framework fit for SRT. The main activities consist out of identification, acquisition, presentation, utilization, and simulation. Except for the simulation activity, we have linked the other activities to the similar named activities in the SOK Framework.

5.1 A Template Method for Improvement of SRT

As we have described in the previous chapter, we divide UPL into HIT and SRT. An improvement in either of these divisions result in an improvement in UPL. Because of the architectural approach of this research, we have defined a method used to improve SRT. Before the architect can measure SRT and identify improvements, they first needs to define the metrics used to measure, analyze, and improve SRT. In the first activity, the identification activity, the architect defines the ACQUISITION CRITERIA, MINING LOGIC, and ACQUISITION LOGIC. The architect defines ACQUISITION CRITERIA by defining the EVENTS and their EVENT MEASUREMENTS. The architect defines the metrics where the application obtains values for. The application stores this data in the form of EVENTS. During the second sub step, for each EVENT the architect defines the measurements which the application obtains values for. The architect uses these EVENT MEASUREMENTS to identify improvements in SRT. Next, the architect defines the MINING LOGIC. The mining logic defines how the application can measure the ACQUISITION CRITERIA so that SOFTWARE OPERATION DATA can be obtained. In the last sub activity in the identification activity, the architect defines the ABSTRACTION LOGIC. The ABSTRACTION LOGIC defines how the architect can abstract the functions and stages from the SOFTWARE OPERATION INFORMATION.

In the first activity, the architect has defined which events the application fires, which metrics the applications measures for these events, how the application obtains values for the metrics, and how the architect can use these events to abstract functions and stages. After the first activity, the architect implements the mining logic in the application and obtains SOFTWARE OPERATION DATA. The architect uses the ABSTRACTION LOGIC to on the SOFTWARE OPERATION DATA to transform the data into SOFTWARE OPERATION INFORMATION. The transformation uses the ABSTRACTION LOGIC to increase the readability of the SOFTWARE OPERATION DATA. Before the architect can make conclusions, the they first verify the correctness of the analysis.

Using various presentation means increases the rate at which information transforms into knowledge. During the third activity, the architect defines presentation means used to visualize the SOFTWARE OPERATION INFORMATION. During the last sub activity, the architect uses the defined PRESENTATION MEANS to create DATA VISUALIZATIONS.

After the SOFTWARE OPERATION INFORMATION has been presented, during the fifth activity, the architect uses these presentations to gain knowledge about SRT. The architect uses the obtained knowledge to identify improvement points of the application. To improve SRT, the architecture defines ARCHITECTURE IMPROVEMENTS. As a result from the architectural changes, the architect changes the software and improves the application.

After the architect has changed the architecture and improved the software, they need to verify these improvements. The architect verifies these improvements by creating a simulation environment in the sixth activity. First, the architect creates a SIMULATION ENVIRONMENT used to simulate SRT on. To obtain SRT measurements, the architect defines a SIMULATION MEANS to simulate users. These simulated users initiate functions which lead to SRT measurements. The architect uses the SIMULATED BEHAVIOR to create a prediction on the effect of architectural changes on SRT.

Software architects can use the described method to measure, improve SRT, and predict the effect of changes in the software’s architecture on SRT. We depicted the template method using a PDD [12]. We have described the PDD notation in section 2.5. The template method consists out of only open concepts and open activities. The method prescribes what activities the architect should perform, and what concepts result from these activities. Appendix C describes the concepts used in the method. The architect uses the template method by instantiating these open concepts and open activities with their equivalent simple or complex activities or concepts. Note that the template method does not have an end point. Because architectural changes can influence the already measured SRT, the architect needs to keep monitoring SRT indefinitely.

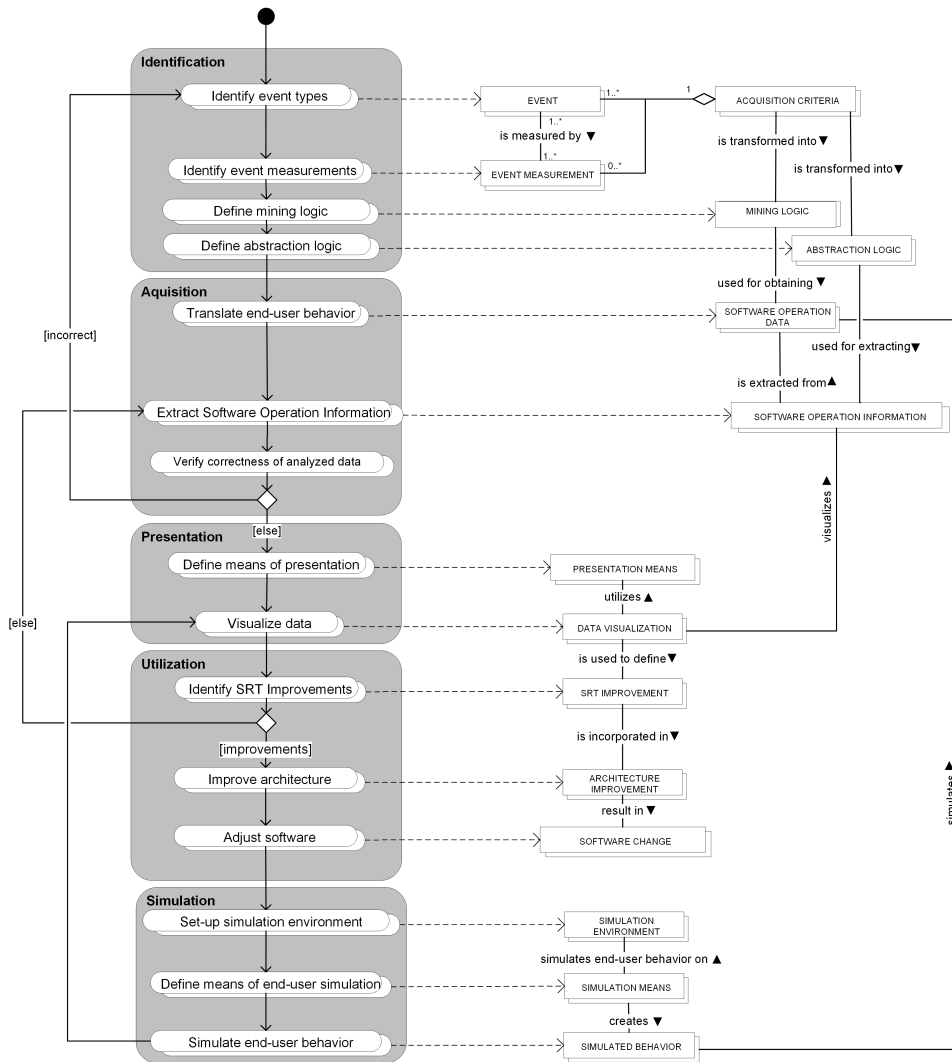


Figure 5.1: The Template Method for Improvement of System Response Time

5.2 Implementation and Supporting Software

To obtain data necessary to obtain knowledge about SRT we propose implementing the architecture depicted in Figure 5.2. In the figure we have used the architecture of the running example and added to the architecture. The addition allows for the implementation of software which raises events, collects the events, and transforms the events. This allows for the architect to obtain knowledge in SRT. The following paragraphs further elaborate on the added components depicted in Figure 5.2.

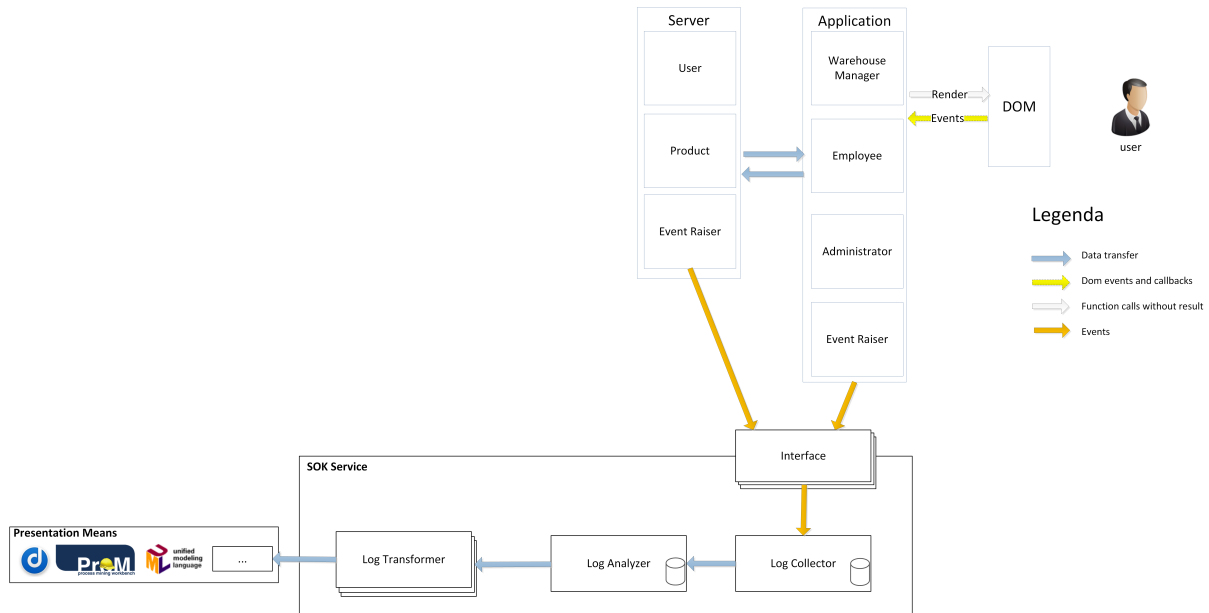


Figure 5.2: Architecture of the Implementation to Support the Method

Event Raiser The architect integrates the EVENT RAISERS in the web application to raise events when they take place. The EVENT RAISERS together must at least have the ability to raise the event types and event measurements defined in the acquisition criteria. When implementing the EVENT RAISER, the architect should be mindful to pollute the application's code as minimal as possible with the code of the EVENT RAISER. As an example, the architect can achieve this on the client by adding functionalities to JavaScript objects provided by the JavaScript engine of the browser e.g. the MutationObserver or the XMLHttpRequest. When the application has access to other application specific objects, the same tactic applies. When the EVENT RAISERS do not have access to from outside of the context, e.g. encapsulated objects or anonymous functions, the EVENT RAISERS must have functionality for the web application to raise custom event types.

Interface The INTERFACE serves as a communication purpose between the EVENT RAISER and the LOG COLLECTOR. Because the architect can implement the EVENT RAISERS at different locations in the application and can write in different programming languages, multiple INTERFACES allow for the most optimal communication.

Log Collector The LOG COLLECTOR has the task to store the events raised by the web application. Depending on active users, type of events, and measurements for each event, the LOG COLLECTOR has to process lots of data. Whilst implementing the LOG COLLECTOR, we see performance as key quality attribute. Therefore, the application needs to process the events as optimal and as fast as possible. Note that for the simplicity of Figure 5.2, we have located the database of the LOG COLLECTOR in the same place as the LOG COLLECTOR but the placement is not required as another placement might increase the efficiency of the LOG COLLECTOR.

Log Analyzer The LOG ANALYZER contains the abstraction logic to transform the events, stored by the LOG COLLECTOR, to the defined stage's and functions. Note that just as with the LOG COLLECTOR the database of the LOG ANALYZER does not have to be at the same location as the LOG COLLECTOR.

Log Transformer When the applications to provide PRESENTATION MEANS are not compatible with the format in which the stages and functions are stored, the architect puts a LOG TRANSFORMER into place. The LOG TRANSFORMER serves as a translator from the log format of the LOG ANALYZER to the format the PRESENTATION MEANS understands. For each type of transformation, the architect puts a different LOG TRANSFORMER into place.

Presentation Means PRESENTATION MEANS are used to present and obtain knowledge from the log of functions and stages. The architect uses the knowledge to identify improvements in the architecture. The PRESENTATION MEANS can thus perform an additional analysis, or serve as a presentation layer. The architect can implement multiple PRESENTATION MEANS programs which interact with each other.

Summarizing, the EVENT RAISERS raise events in the client and in the server. The application stores these events using the LOG COLLECTOR. The EVENT RAISERS communicate with the LOG COLLECTOR through (multiple) INTERFACES. The LOG ANALYZER analyses the events into functions and stages. These functions and stages are transformed into a specific format by the LOG TRANSFORMERS. The PRESENTATION MEANS define the specific format the functions and stages are transformed into. The PRESENTATION MEANS use the transformed functions and/or stages to make visualizations.

5.3 Identification of Software Operation Knowledge

During the software operation knowledge identification phase of the method, the architect defines the acquisition criteria, mining logic, and abstraction logic. The acquisition logic defines what metrics the application obtains values for. The mining logic defines how and where the application obtains the values for the defined metrics. Abstraction logic describes the manipulations needed to analyze the metric data.

5.3.1 Acquisition criteria

During this activity, the architect defines the metrics which the application measures. The architect uses these measurements in the analysis. To define the acquisition criteria, the architect defines the execution data needed to obtain information about SRT. To analyze the data, we propose to structure the storage of this data to allow for process mining. The structure we propose structures the data using the XES Meta model [34], see Figure 5.3. We structure the data using the XES format because XES is the de facto standard for process mining. XES uses an XML based syntax used for process mining purposes. An XES file contains a single LOG, containing information about the measured process. The LOG can contain TRACES containing information about one specific occurrence of a process. The LOG, TRACE, and EVENT can have ATTRIBUTES which in their turn may contain ATTRIBUTES. ATTRIBUTES provide information about a LOG, TRACE, and EVENT. A TRACE can contain EVENTS which provide with information about an activity that took place in a TRACE, e.g. a change in the DOM or a user click in the DOM. Note that the EVENT on its own does not provide any information. The ATTRIBUTES provide with information about a LOG, TRACE, and EVENT. A CLASSIFIER classifies ATTRIBUTES. Comparing the ATTRIBUTES of a CLASSIFIER of two EVENTS determine whether one can consider two EVENTS equal for that particular the CLASSIFIER. To provide ATTRIBUTES with semantics, one can include EXTENSIONS. EXTENSIONS thus give more meaning to the value of an ATTRIBUTE, e.g. a Time extension gives more meaning to the time format of a Date Attribute [34].

Because we measure SRT using multiple metrics, we can specify multiple event types, and have an analysis step to functions and stages, the XES Meta model alone does not suffice. Therefore we define a model specific for measuring SRT based on the XES Meta model. When defining the acquisition criteria, the architect can use Figure 5.4. The figure uses an UML class diagram to depict the structure which the architect can use when defining the acquisition logic. In the figure, the components of SRT have a white color and the XES meta model components have a gray color. The ACTIVITY, as used in process mining, serves as a base to provide with information about SRT. The ACTIVITY and can be, but is not limited to,

a FUNCTION or a STAGE. Both the ACTIVITY and EVENT have multiple ATTRIBUTES. The LIFECYCLE represents the current state e.g. schedule, start, or complete. The TIMESTAMP indicates when a certain ACTIVITY or EVENT has occurred. An INSTANCE gives information about which ACTIVITIES or EVENTS are related to each other. The NAME gives more meaning to the ACTIVITY or EVENT.

The architect utilizes the figure by defining the EVENT TYPES, which together measure SRT, and defining the METRICS for each EVENT TYPE. Using the running example introduced in chapter 3, we have described an example set of EVENT TYPES and METRICS for each EVENT TYPE in Table 5.1. We utilize the metrics in the following sections to provide the architect with information about SRT.

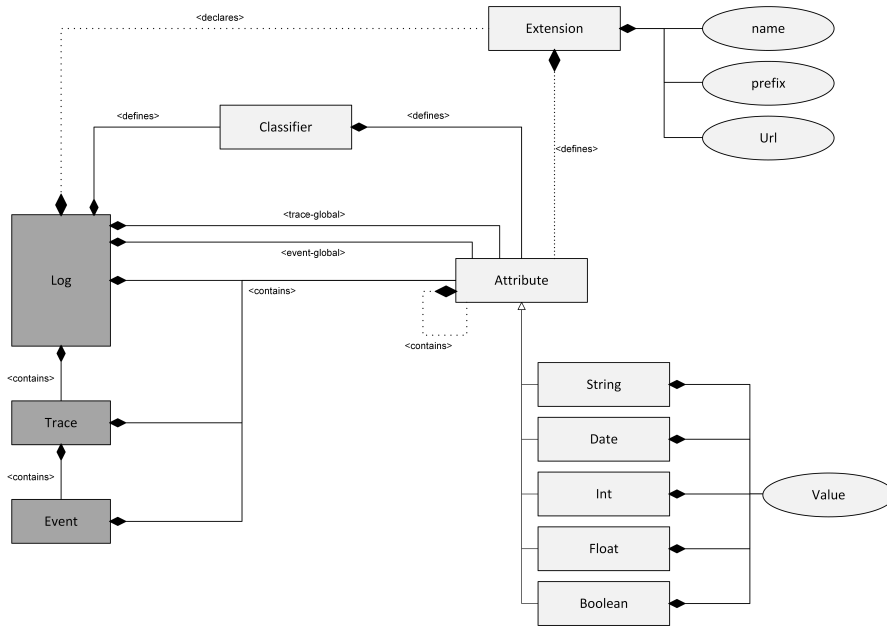


Figure 5.3: Meta Model of XES expressed by means of an UML Class Diagram [34]

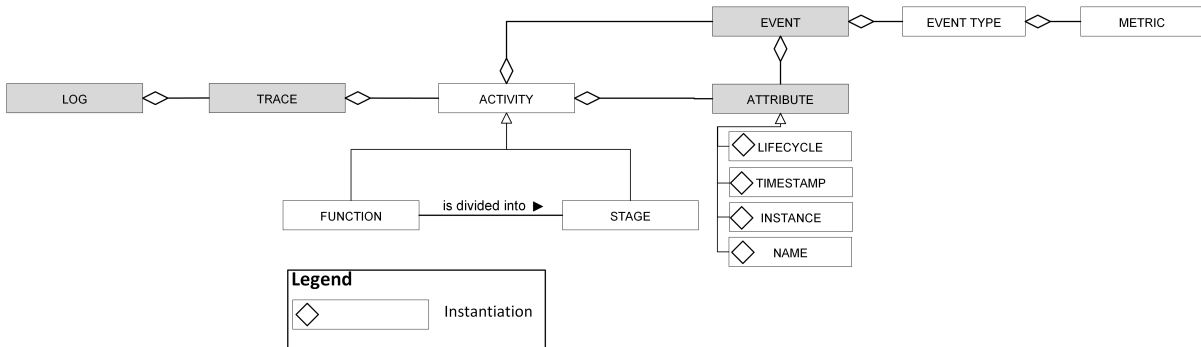


Figure 5.4: Relation Between SRT Components and XES Components.

Measurement	DOM Nodes Added	DOM Nodes Deleted	Target	Request Length	Url	Type
Event						
DOM Manipulation	✓	✓			✓	
Ajax			✓	✓	✓	
DOM Event			✓		✓	✓

Table 5.1: Example Measurement for Events

5.3.2 Mining Logic

The architect can locate and measure the events used to gain knowledge about SRT of web applications at least one client, at least one server, and the user. After the architect has defined the events, they need to define where to obtain these events from (e.g. which (sub)system and which component) and how application measures the event.

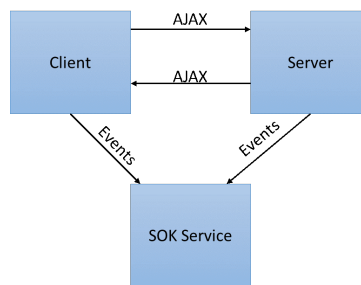


Figure 5.5: Interaction between Client, Server, and SOK service

To store the gathered events, the architect can choose to give responsibility to one of the clients or one of the servers to gather and store SRT data from the client(s), server(s), and users of the application. The architect can also provide the client(s) and server(s) access to the same database to store SRT data separately. We find the second solution the most performance efficient because the application has no overhead when storing the data e.g. the application stores the data directly into the database instead of being sent through (multiple) other clients or servers before storing. We suggest the approach to create a webservice accessible for the client(s) and the server(s), see Figure 5.5. We name the webservice used for storing events the SOK service. The research focuses on the link between the client and the SOK service.

Looking at the running example, the SOK service fits in the architecture as depicted in Figure 5.2. The architecture of the running example has a client and a server. On both the client and server, events can occur. When an event occurs, the application catches the event e.g. a JavaScript program on the client or a program on the server. The application sends the caught events to the SOK service using AJAX on the client and a web request on the server which then stores these events into a database. In section 5.2 we have provided with additional information on the SOK service.

5.3.3 Abstraction Logic

After the architect has identified the events have and has defined where and how the application measures the events, a log existing of events is obtained. As we have stated in the previous chapter, a function is a measurement of SRT. Multiple stages can divide a function. To gain information about SRT, we first need to transform the event log into functions and stages.

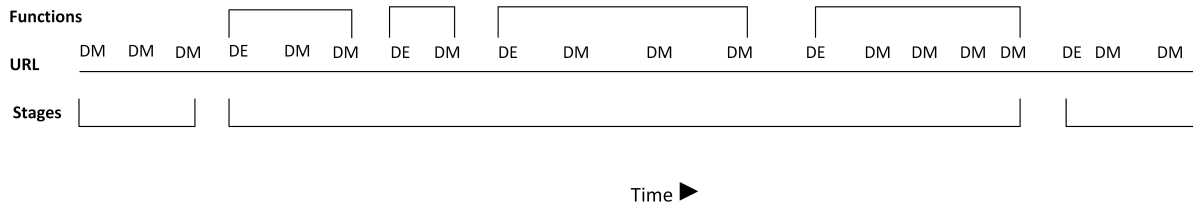


Figure 5.6: Example event timeline occurring on a single URL. The timeline contains DOM Manipulation events (DM) and DOM events (DE)

Defining functions and stages After an event log has been abstracted, the architect needs to define logic used to obtain a function log and a stage log from the event log. Using the running example, we provide with a partial event log in Table 5.2.

Looking at Figure 5.6, we have abbreviated the DOM Manipulation (DM) and DOM event (DE). The DOM event represents user interaction and the DOM Manipulation represents the application responding to an interaction. The figure shows the DOM Manipulation events and DOM events on a single URL. Looking at the figure we define three stages. In the first stage, the application uses DOM Manipulation to present the user with the page. In the second stage, the user and application are interacting. During the third stage, the user requests to view another page. Before the application loads the requested page, the application cleans up the current page. Throughout these stages we see multiple functions occurring. A function starts with a user action. The function ends when the application has used (possibly) multiple DOM Manipulations to respond to the user action. When the application responds to the user, we see no further user interaction in the meantime.

As we have described in the previous paragraph, we define that a function always starts with a user action (measured with a DOM event). The application responds to the action by changing the DOM (measured with DOM Manipulation events). During a function we see no additional user actions. Using the following statement we can define the logic to abstract functions: $\langle \text{DE}; \text{DM}^* \rangle$. When applying the logic on Table 5.2, we sort the list on timestamp in ascending order. Then we look for the first occurrence of a DOM event which is event #3. Then we include all following DOM Manipulation events (#4) until the next DOM event (#5). The analyzed function consists out of events #3 and #4. We see event #5 as the start event of the next function which consists out of events #5 and #6. Note that after event #7 there are no other DOM events. Therefore, we cannot analyze events #7 and #8 into a function as the function could be incomplete. We stated an extended event log together with the analyzed functions in Appendix D.

Using the following statement we can define the logic to abstract the three stages: $\langle \text{DM}^* \rangle$, $(\langle \text{DE}; \text{DM}^* \rangle)^*$, $\langle \text{DE}; \text{DM}^* \rangle$. In the first stage, we can observe DOM Manipulations prior to the first DOM event ($\langle \text{DM}^* \rangle$). During the second stage, the user can perform functions $(\langle \text{DE}; \text{DM}^* \rangle)^*$. The last user action (DOM event) on a URL is the start of the page cleanup stage. In the last stage, the application can use DOM Manipulations to clean up the page ($\langle \text{DE}; \text{DM}^* \rangle$). Looking at Table 5.2 we sort the list in ascending order on the timestamp. Looking at the table, we identify all three stages. For the first stage $(\langle \text{DM}^* \rangle)^*$ we include the DOM Manipulation events before the first DOM event. We analyze events #1 and #2 as the first stage. The second stage $(\langle \text{DE}; \text{DM}^* \rangle)^*$ starts with the first event of the first function performed by the user (DOM event). Looking at the table, we see event #3 as the first DOM event on the URL. As described in the previous paragraph, the last function on the URL ends with event #6. We analyze events #3, #4, #5, and #6 as the second stage. The last stage starts with the last DOM event on a URL and could include following DOM Manipulations. We see event #7 as the last DOM event on the URL. We see event #8 as the following DOM Manipulation. We analyze events #7 and #8 as the third stage. Appendix D presents an extended event log together with the analyzed stages.

#	Timestamp	Event Type
1	1375642943	DOM Manipulation
2	1375642951	DOM Manipulation
3	1375642962	DOM Event
4	1375642973	DOM Manipulation
5	1375642984	DOM Event
6	1375642988	DOM Manipulation
7	1375642993	DOM Event
8	1375643033	DOM Manipulation

Table 5.2: Partial Example Event Log

5.4 Acquisition of Software Operation Knowledge

When the architect has defined and implemented the acquisition criteria, mining logic, and abstraction logic, the implementation measures and stores software operation data. After we have obtained the data, the implementation transforms the data into a usable format fit for data mining and process mining.

5.4.1 Data Mining and Process Mining

Bramer describes data mining as an activity performed in knowledge discovery [35]. Data mining allows finding rules, or patterns out of a set of data. Data mining transforms the dataset into an understandable and usable structure [35].

After data mining has been performed and the architect has found rules and patterns, they put the process mining techniques into practice, to model processes [36]. These modeled processes provide with information used in process discovery, process conformance, and process enhancement. To create these process models, process mining uses an available process log. The process log has to have information about at least the order of events that took place. When a minimal process log can be obtained, process mining techniques and algorithms can be applied on virtually any process [36] [37].

As stated in section 5.3.1, a link between XES, process mining, and SRT has been made. The obtained data through the abstraction logic is therefore perfectly adaptable to use in process mining and by process mining tools like ProM Tools¹. Example process mining techniques include: heuristics mining, fuzzy model mining, dotted chart analysis, and more.

5.4.2 Verify Correctness of Information

After the architect has implemented the mining logic into the application to obtain software operation data, and abstraction logic has been performed to analyze the data, the architect needs to verify the analyzed data for correctness. After we have analyzed the entire event log from the running example, we grouped the functions and stages together on their name and URL. We have calculated the duration by

¹www.promtools.org

subtracting the timestamp of the last event from the timestamp of the first event withing a particular function or stage.

An indication for an error in the results is a higher than expected duration. The standard deviation provides with another indication. The architect utilizes the standard deviation to see the dispersion of the measurements. A high standard deviation indicates a high variation in the measurements, meaning that there is a high chance of an error in the acquisition criteria, mining logic, or/and acquisition logic. A box-plot can visualize the dispersion of durations and aid in the identification of outliers. Figure 5.7 depicts the SRT measurements of the "ship product" function from the running example. In the box-plot we used the interquartile range of 1,5 to identify outliers. In other words, we consider every measurement with a duration above or below the 1,5 interquartile range as an outlier. The figure depicts the outliers with a circle. Looking at the figure, we cannot identify any outliers lower than the 1,5 interquartile range. We identify 51 outliers higher than the 1,5 interquartile range. The outliers start with a duration of 324 milliseconds and end with a duration of 499 milliseconds. After identifying these 51 outliers, the architect needs to look into the containing events of these measurements. As a result, the architect can identify the reason for the duration of these measurements. The architect can use these reasons to improve the acquisition criteria, mining logic, and abstraction logic. When the architect does not find improvements whilst looking into these outliers, we consider these outliers as correct measurements. The architect can use these outliers to improve the architecture.

When the results indicate incorrect results, the architect should look into the list of containing events from the functions and stages and try to replicate the results. Reasons for an incorrect result for functions include additional events, missing events, missing event types, and/or incorrect event types. When the results indicate errors in functions or stages, the architect must redefine the acquisition criteria, mining logic, and abstraction logic.

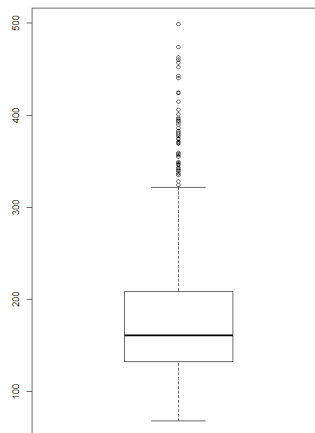


Figure 5.7: Box-plot of the running example's "ship product" function, the duration in milliseconds is plotted on the y axis.

5.5 Presentation of Software Operation Knowledge

After the architect has analyzed the data, they need to visualize the data (transforming the data into information). When architect visualizes the data, they can interpret the information correctly, and use this information to gain knowledge. The following paragraphs describe multiple means to visualize the data so that knowledge in SRT can be obtained. Note that the following presentation means are just a couple of examples out of the inexhaustible list of possibilities. To base a presentation means on, we use the EVENT, EVENT TYPE, ATTRIBUTE, FUNCTION, AND STAGE components as described in Figure 5.4.

Most process models are based on the assumptions that process logs are reliable and trustworthy, and that the logs reflect an exact process. However, in a real world situation, these assumptions are mostly incorrect and result in a so called spaghetti model [38]. Figure 5.8 depicts the spaghetti model

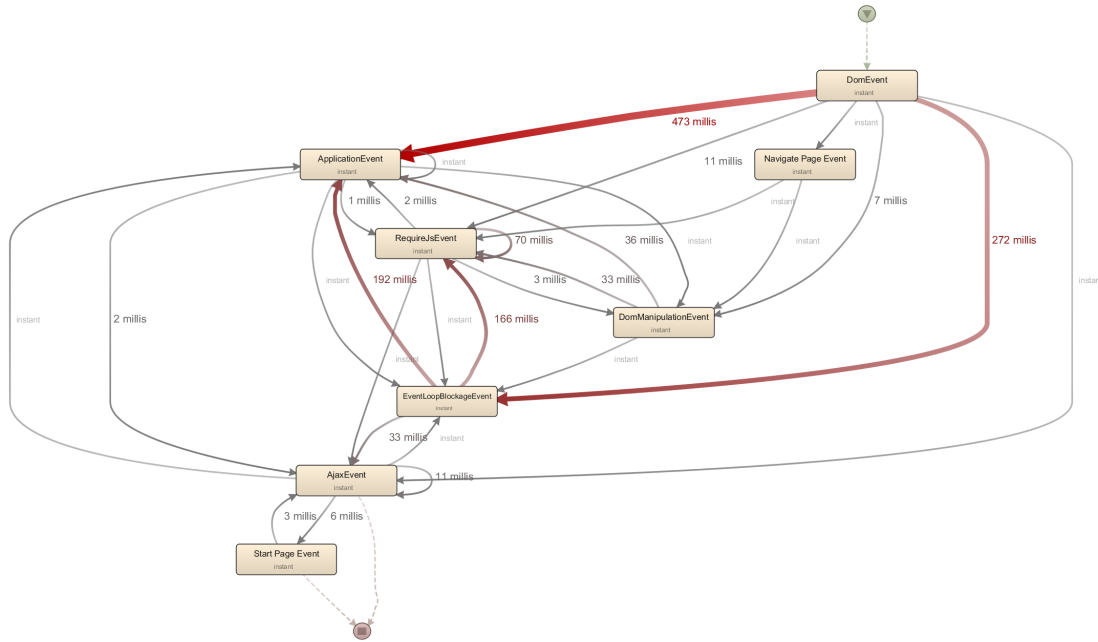


Figure 5.8: Model of the Running Example's Function Log

of the "ship product" function of the running example described in Chapter 3. The spaghetti model, depicts the entire log with every possible path in the log. However, the architect cannot use the model complete because the model is too large and therefore confusing, and obscure. The *fuzzy model* uses conflict resolution in binary relations, edge filtering, node aggregation, and node abstraction to create a high level mapping of the process using the log [38]. The resulting *fuzzy model* is more readable and understandable than the spaghetti model. However, the resulting model does not depict every case. Therefore the architect must use caution when making conclusions based on this model. The architect can efficiently use this modeling technique when there the log contains a lot of variance. Applying the fuzzy modeling technique on Figure 5.8 resulted in Figure 5.9. Note, to magnify the effect of the fuzzy modeling technique we have added additional event types to the log. In Figure 5.9 we have depicted the duration between events on the arrows in milliseconds. The arrow size represents the duration in retrospect to the other durations e.g. a thick arrow means a high duration compared to the other durations.

The architect can apply the Heuristic Miner on a process log which contains little variation in events. The architect can use the heuristics mining approach in three steps where the first creates a dependency/frequency table, the second uses the dependency/frequency table to construct a graph, the third step uses these tables and graphs to reconstruct a workflow/frequency-net [39]. Applying the heuristics mining technique on the stage log of the running example, we have obtained a workflow/frequency-net, see Figure 5.10. The architect gathers knowledge from this model in the same way as the Fuzzy model.

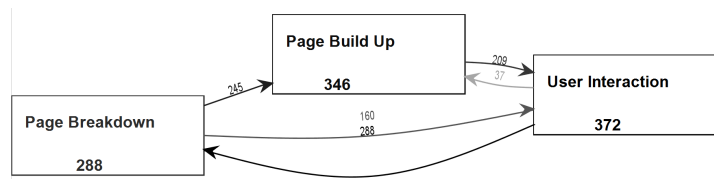


Figure 5.10: Heuristics model of the Running Example's Stage Log

The architect can make a dotted chart analysis to see patterns in a process log. We have plotted the cases on the y-axis and have color coded the events types, ordered the events types by occurrence,

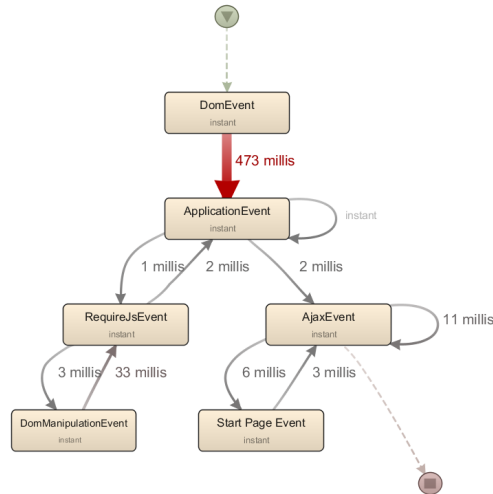


Figure 5.9: Fuzzy model of the Running Example's Function Log

and plotted the events types as dots on the x-axis [40]. Figure 5.11 depicts the dotted chart analysis obtained from the running example's function log. The architect can use the patterns in the dotted chart to compare functions for (partial) equality. When the same pattern in event types arises, the functions are likely to be (partially) the same.

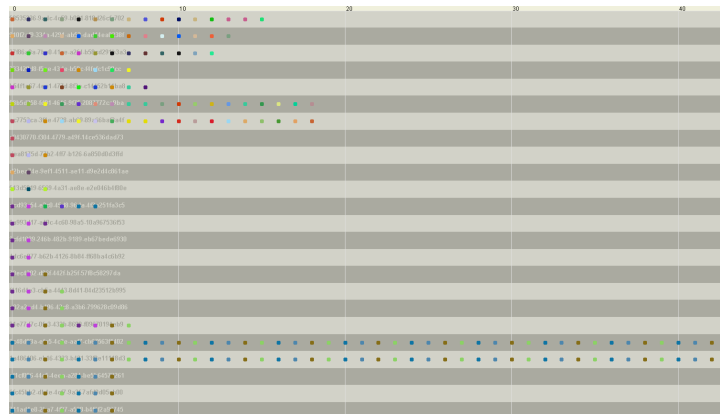


Figure 5.11: Dotted chart of the Running Example's Function Log

5.6 Utilization of Software Operation Knowledge

After the data has been collected, analyzed, and visualized, the architect needs to use the provided information. This section describes how the architect can put the data and presentation means into practice to improve into SRT. In the following paragraphs, we utilize the figures described in the previous section.

Looking at Figure 5.9, the measured durations between events are between 0 milliseconds and 33 milliseconds except the 473 milliseconds duration between the DOM event and the ApplicationEvent. Therefore we can identify an improvement point between the DOM event and the ApplicationEvent.

As one can see in Figure 5.10, the figure depicts the flow which the architect expects (page buildup to user interaction to page breakdown). However, the figure also depicts other flows. We can conclude from the connection between page breakdown and user interaction that URLs do not have a page buildup stage. We can conclude from the connection between user interaction and page buildup that URLs

do not have a page breakdown stage. The lack of a connection from page buildup to page breakdown leads to the conclusion that URLs have a user interaction stage. URLs which do not include a page breakdown stage might indicate that the application does not clean up the page on the URL properly. URLs without a page buildup stage imply that the application does not alter the DOM when the page opens. This indicates that the user action leading to the change in URL did not have any effect which might indicate an unresponsive link.

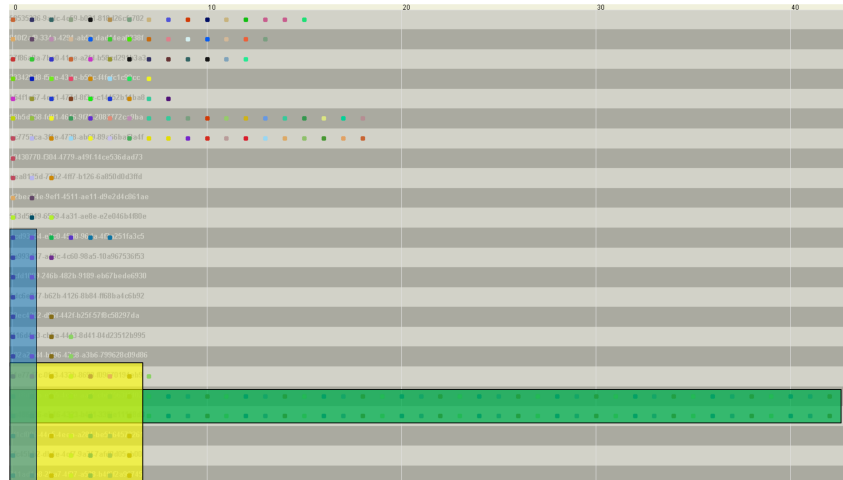


Figure 5.12: Dotted chart of the Running Example's Function Log

Looking at Figure 5.12 we have identified similarities with color coded overlays. We have identified 2 functions with exactly the same containing events (the green overlay). We have identified 6 functions which start with the same 7 events (the yellow overlay). And we have identified 13 function which start with the same 2 events. From this visualization we have identified which functions we assume similar and which we assume can group together. Note that with a large dataset of functions and containing events, we assume that the dotted chart analysis becomes to big to read. Therefore the architect can only use this chart in small datasets.

5.7 System Response Time Simulation

As a result from the previous action, the architect has obtained of improvements in the architecture to improve SRT. Before the architect can improve the architecture, they need to set up a simulation environment. The architect first needs to verify the simulated environment by checking whether the environment can provide with SRT measures comparable to the obtained SRT from the production environment.

First, the architect sets up a simulation environment by installing a replica of the application used in the method. Next, the architect needs to define what a typical user does on the application. Using the function log of the running example, we have created a fuzzy model depicted in Figure 5.13. In the figure the architect can see which functions the user has mostly executed after each other. Using the figure, the architect can set up a use cases representative for the usage of a normal user. From the figure we can abstract the following use cases: "Person Edit, Remove Rights, Add Rights, Remove Rights, Person Edit", "Person Edit, Remove Rights, Organization Add, Remove Rights, Person Edit", "Person Edit", or "Person Edit, Remove Rights, Person Edit".

After the architect has created the simulation environment and has defined use cases which represent the usage of a typical user, the architect needs to simulate SRT onto the environment. To perform the simulation, the architect can chose to have a team perform these use cases by hand. The architect can also choose to simulate these use cases by implementing and/or using a program which records a set of actions and repeats these actions. Using this method, the architect only has to perform the use case once and the program repeats the use case. Lastly, the architect can choose to create and implement an application which uses a set of predefined actions representing a use case.

After the architect has finished the simulation and the verification of the simulation, they use the same techniques as described in the method to obtain the similar figures and tables. Comparing these figures and tables indicates whether the architect has set up a representative simulation environment. After the architect has created a representative simulation environment they can implement the obtained improvements in the architecture. After the architecture has changed and the application has been updated accordingly, the architect uses the same techniques as described in the previous paragraphs to simulate the user. When the architect uses the same modeling techniques to create the same models as before, they can compare these figures to indicate the effect from these architectural changes.

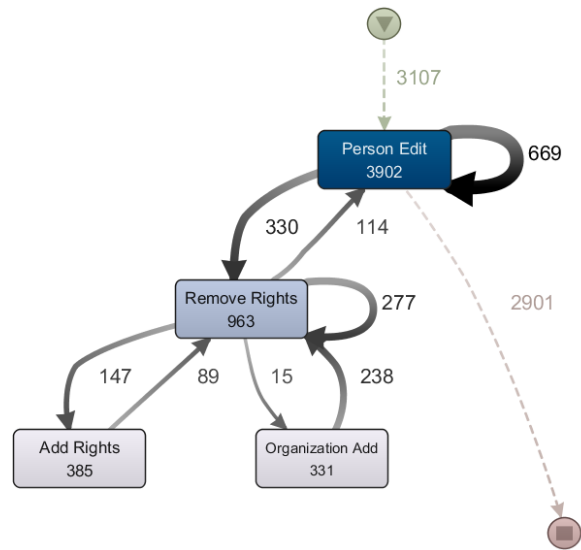


Figure 5.13: Fuzzy Model of the Running Examples Function Log

To validate the method for improving user-perceived latency as described in the previous chapter, this chapter describes a case study performed at AFAS Software B.V.. The case study was performed during a 4 month period from the beginning of January 2014 until the end of April 2014. We validate the method described in the previous chapter by utilizing the method to measure, analyze, and identify improvements in system response time (SRT). AFAS Software B.V. is currently developing a new web application, internally named Profit Next. Because the application's logic rapidly increases, the user experience varies for each piece of logic implemented. We see that the architects of AFAS Software B.V. need a structured approach aiding them in implementing an automated solution to gain knowledge about SRT. The architects at AFAS use the automated solution for setting a baseline for SRT. The architects use the baseline to keep a grip on SRT, making sure the additional implemented logic does not result in an unacceptable increase in the duration of SRT. Profit Next is therefore a good candidate to validate the method. The following sections describe the company AFAS Software B.V., Profit Next, and the instantiation of the method.

6.1 Case Company AFAS Software B.V.

AFAS Software B.V. is a Dutch organization focused on building enterprise software. AFAS was founded in 1996 after a management buy-out from Getronics. The headquarters of AFAS is based in Leusden, the Netherlands. The international offices of AFAS Software B.V. are located in Belgium and Curacao. At the time of writing this thesis AFAS employs over 300 people. The vision of AFAS is to automate administrative business processes. To realize its vision, AFAS focuses on building and retailing an integrated enterprise software product for both small, and large organizations in the accounting sector, welfare sector, and the education sector. In 2013, AFAS obtained a profit of over 22 million. AFAS has two products, Profit 2014 and AFAS Online.

Profit 2014 incorporates customer relationship management, course management, document management, financial, fiscal, human resources, logistics, payroll, projects, workflowmanagement, an intranet, a portal, and a website. Because these modules are integrated into one product, the administrative processes that encompass one or multiple modules can be automated to the fullest extent. At the writing of this thesis, Profit 2014 has over 9.400 customers and over 118.000 users are working with Profit 2014. Customers of AFAS include Vakantieveilingen, Connection, Jumbo, OSGMetrium, CoolBlue, Staatsloterij, and Pon Dealer Group. AFAS Online is an online application which can be used by individuals to manage their household expenses. AFAS Online currently has over 172.000 users.

The product is built for Windows and uses Visual Basic 6 and for add-ons .NET C # 4.0. These are still the same technologies as were in use in 1999. To keep innovating their products, AFAS has initiated the development of an application internally called Profit Next. Profit Next, is a complete rebuild of Profit 2014.

6.1.1 Profit Next

To keep innovating, AFAS is currently working on Profit Next. Profit Next uses a model-driven solution where the model is completely separated from implementation and technology. Because of the separation, the technology put into practice to implement the model is easier to change and updated. The model-driven approach thus allows for indefinite reuse of the model. The separation of technology and model, allows the architects to automatically add components with ease. Therefore, we can easily implement a module used to measure, analyze, and improve SRT. Profit Next encompasses a studio environment, in which such a model is defined, and a generator which uses the model (application definition) generated by the studio to create the application.

Using the application definition, the generator creates the functionality of the Profit Next application. The Profit Next application has a web based client and server. Both components are located in the cloud. The components of the client and their interaction have the focus when improving SRT and are depicted in Figure 6.1. The client uses a tree structure of Components. These Components are built according to the architectural pattern MCV and use the JavaScript based Application of Profit Next to operate. The Components and Application use the jQuery framework for rendering to the DOM. The Server Communication of the Application takes care of the communication with (pre) processing to and from the server. The application performs communication using AJAX (through the jQuery framework).

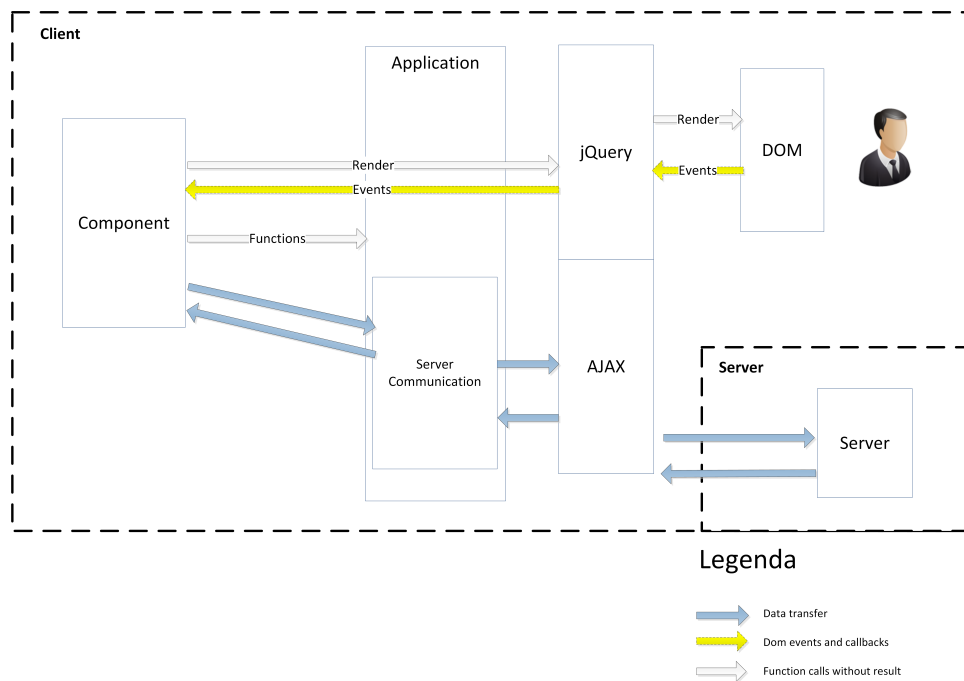


Figure 6.1: Simplified Architecture of the Profit Next Client

6.1.2 Case Study Description

To validate the method presented in the previous section we instantiated the method and performed a case study at AFAS Software B.V. using the method. During the case study and together with the architects at AFAS, we identified the events and metrics needed to provide them with a structured approach to measure, analyze, and improve SRT. Next, we implemented a logging module in Profit Next which gathers the defined events and metrics. Because Profit Next is still in development, users are not yet working with the application. Therefore in consult with the architects, we defined use cases representing the usage of a regular user. To simulate the usage of the application, we created and implemented a simulation module which we used to automatically perform the defined use cases multiple times. After we gathered the data generated by the logging module we defined and used multiple presentation means. In consult with the architects at AFAS, we used these visualizations to gain knowledge about SRT. Based on the results, we provided the architects at AFAS with recommendations to improve SRT.

6.2 Case Study Structure

To obtain software operation knowledge (SOK) on SRT, we implemented a logging module in Profit Next. The logging module fires and stores events from the client of Profit Next. Using on the logging module, we obtained software operation data. The following subsections elaborate more in the implementation of the logging module, the fired events, and the gathered data.

6.2.1 Structure of the Architecture

To measure, analyze, and improve SRT, we implemented the SOK service as described in section 5.2 and written a JavaScript API. We implemented these components in Profit Next as depicted in Figure 6.2. Note that the server can also use the same SOK service. However we do not further elaborate upon this usage because it does not have the focus. The following paragraphs elaborate more on the implemented components.

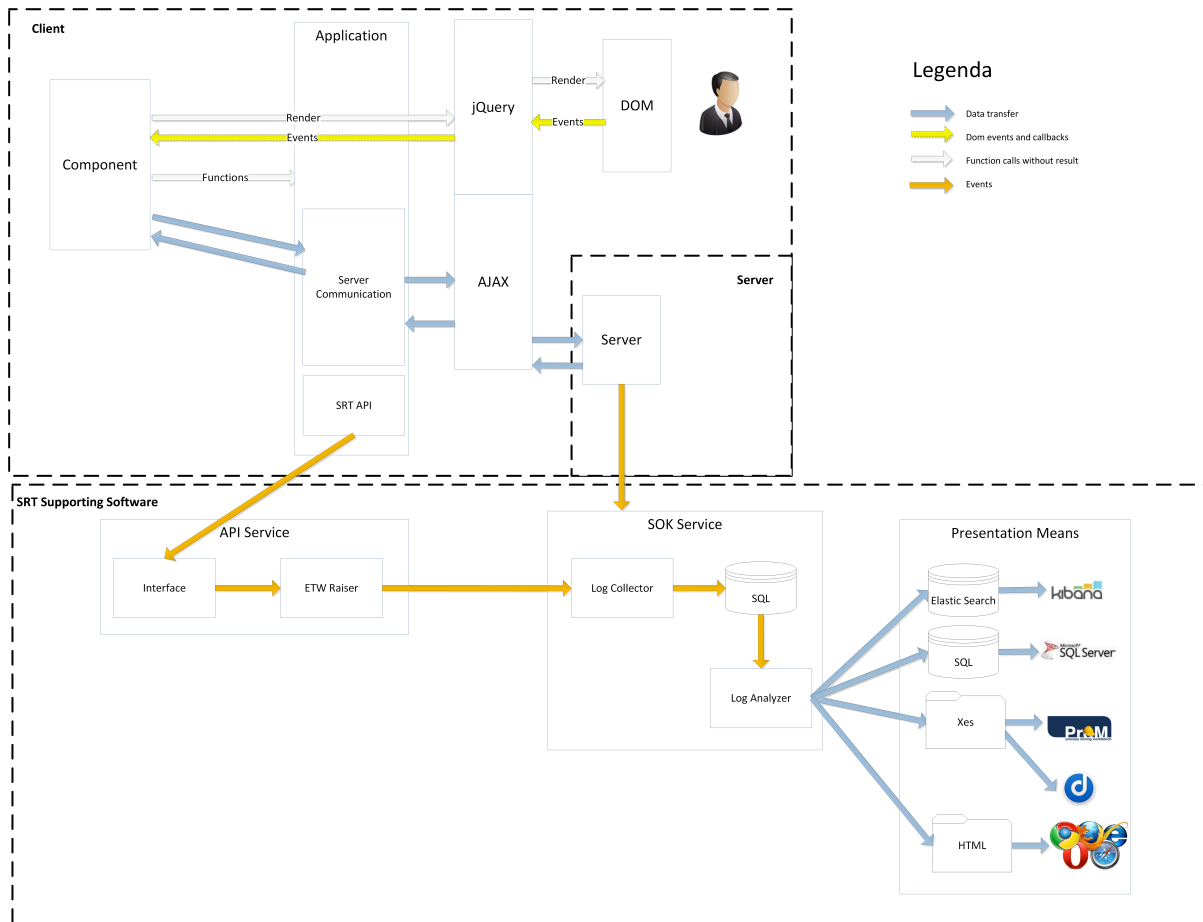


Figure 6.2: Architecture of the Profit Next Client With SOK Service

SRT API We implemented the SRT API as a JavaScript API and included the API into the application. We choose to implement a JavaScript API because the API can access global JavaScript objects and dynamically add code to these objects. Using this approach, we minimized amount of code polluted with the code of the API. After the architect has included the script, they can instantiate a new acquisition object using the options. These options indicate whether the API should fire the ACQUISITION, AJAX, EVENTLOOP, DOM, DOM MANIPULATION, and/or REQUIREJS events. The acquisition object has four methods which can stop the acquisition, instrument an object, send the events in queue to the API SERVICE, and fire a custom event. The instrumentation functionality of the acquisition object surrounds every function of the provided JavaScript object with a start and end event. When the queue of events

contains 100 events or the application has not fired an event for 100 milliseconds the acquisition object automatically sends the events to the API SERVICE.

API Service The API Service encompasses an INTERFACE where the SRT API to sends the collected events to. We choose to implement the API Service so that we can store the events raised by the SRT API on multiple clients in a single place. Because we do not have to take into account gathering and combining events from multiple places, having the events in a single place, makes the analysis easier. The API Service further encompasses a ETW RAISER which converts the obtained events to ETW (Event Tracing for Windows) events and raises these events. When we activate the listener, the ETW mechanism stores these ETW events inside a queue and processes them. Advantages of this approach include:

- A minimal waiting time for the client because the client does not have to wait for the application to process the events;
- We can easily turn on and off the ETW mechanism because by default, ETW events are not raised when we have no listener set in place;
- Microsoft designed and built the ETW mechanism for fast event handling.

The ETW mechanism has a disadvantage. When the application pushes events to a full ETW queue, the ETW mechanism drops these events. Therefore we cannot guarantee that the mechanism stores every event. Because of the time constraints of the research and because we can use techniques to identify missing events, we choose to use the ETW mechanism. As we described in the previous chapter, we can use a box-plot to visualize the outliers. We label a measurement as an outlier when we obtain a duration of 1,5 times above or below the interquartile range.

SOK Service The SOK service contains a LOG COLLECTOR, a SQL database, and a LOG ANALYZER. We developed the LOG COLLECTOR to listen for the ETW events raised by the ETW RAISER and to store these events into the SQL database. Note that although we depicted the API Service and SOK Service as separate components, for the ETW mechanism to work they should be located onto the same physical machine. We choose to implement the SOK Service as described so that we can easily turn on and off the logging mechanism on the SOK Service, Presentation Means, and on the ETW RAISER. Simply turning the SOK Service on and off turns the prior described logging modules on and off.

Presentation Means The LOG ANALYZER contains logic used to extract the stages and functions from events stored in the SQL database. After the application applies the logic onto the set of events, the LOG ANALYZER stores the abstracted stages and functions into an ELASTIC SEARCH database, a SQL database, XML files using the XES format, and HTML files. To visualize the data in a Kibana dashboard¹ we use the compatible ELASTICSEARCH² database. For querying the dataset, we use a SQL database. The XML files in XES format allow for compatibility with ProM Tools³ and Disco⁴. We use these tools to perform various process mining techniques and visualizations on. The HTML files are shown in a browser to visualize the stages, functions, and processes of a single user session on a time-line.

6.2.2 Measurements

During the first phase of the method named the identification phase, we define the acquisition criteria, mining logic, and abstraction logic. To identify the acquisition criteria, in consultation with the architect, we used Figure 5.4 to define the event types and metrics, see Table 6.1. The following paragraphs provide with an elaboration on these event types. Table 6.3 provides with a description for the metrics. Table 6.2 provides with a description for the attributes.

¹www.elasticsearch.org/overview/kibana/

²www.elasticsearch.org

³www.promtools.org/

⁴<http://www.fluxicon.com/disco/>

Event Type(Name)	Attributes							Metrics											
	Life-Cycle	URL	Origin	Type	Instance	Session	User Agent	User Id	HeapSize	HeapSizeLimit	Timestamp	Added	Modified	Deleted	Width	Height	MaxWidth	MaxHeight	Blockage
AJAX	✓	✓	✓		✓	✓			✓		✓								
DOM		✓	✓	✓		✓			✓		✓								
DOM Manipulation		✓				✓			✓		✓	✓	✓	✓					
Exception		✓		✓		✓			✓		✓								
Application	✓	✓	✓	✓	✓	✓		✓		✓									
RequireJs	✓	✓			✓	✓			✓		✓								
EventLoop Blockage		✓				✓			✓		✓								✓
Acquisition	✓	✓				✓	✓		✓	✓	✓				✓	✓	✓	✓	
User		✓				✓	✓	✓	✓		✓								

Table 6.1: Event Types and Their Metrics and Attributes

Acquisition Event Type The ACQUISITION event gives information about when the logging mechanism has been initiated on the client (life-cycle start) and when the logging mechanism has stopped (life-cycle complete). Usually the logging mechanism starts when the user opens the application in the browser and ends when the user closes the browser window. To measure type of browsers interacting with the application, the acquisition event with life-cycle start contains information about the browser, the user agent (which contains information about the version number, operating system, and device type), the available screen size relative to the actual screen size, and the available and used JavaScript heap size. As an example, when the user starts the Profit Next application, the application fires an event with the ACQUISITION type and a the life-cycle start. When the user closes the browser window, the application raises an event with the ACQUISITION type and a life-cycle complete.

DOM Manipulation Event Type Profit Next uses DOM manipulation to visualize the result of the user’s action to the user. We measure the DOM MANIPULATION events using the JavaScript Mutation Observer ⁵ and contains information about in the affected DOM nodes together with the type of manipulation e.g. add, remove, or alter. The Mutation Observer gathers a list with the DOM nodes affected by DOM Manipulations and fires the events when other JavaScript code does not use the eventloop. Note that because the Mutation Observer includes a mechanism to not block the eventloop, the time the event occurs and the application registers the event might differ.

AJAX Event Type To measure the usage of the communication the application fires an event when the application initiates an AJAX call (life-cycle schedule), sends an AJAX call (life-cycle start), and when the application receives a response (life-cycle complete). When the application sends an AJAX call,

⁵<https://developer.mozilla.org/en/docs/Web/API/MutationObserver>

the application stores information about the transmitted data, size, and when the application receives a response, the application stores information about the response size.

Eventloop Event Type When the application executes one or more processor intensive JavaScript functions, the functions can block the eventloop and postpone the execution of other JavaScript functionalities. The application measures every 50 millisecond whether or not the JavaScript blocks the EVENTLOOP. We consider the eventloop as blocked when we measure a difference of 50 milliseconds between the expected and the actual execution time.

DOM Event Type When the user interacts with the application, the application fires DOM events e.g. a left mouse click, right mouse click, mouse hover, keyboard button press or keyboard button release. For the case study, the application fires a DOM event when the application registers a click event on a link, button, or span element in the DOM. The application also fires a DOM event when an input field or text area gains or loses focus.

Error Event Type Whenever an error occurs whilst using the application, the application stores the error message and error location in the ERROR event.

User Event Type Multiple users can concurrently use Profit Next. The application identifies the user after a login and recognized the user by a username. We choose to collect the username in a separate event type instead of an attribute because when the application fires an event with one of the other event types, we cannot ensure that the user has already logged in. When the user has not logged in we do not know the username, and thus we cannot ensure that we store each username. Therefore we fire an event with the USER event type after the user has logged into the application. The USER event contains a username.

RequireJs Event Type Profit Next uses the RequireJs framework ⁶ to load necessary modules for the application to run. The application fires the REQUIREJS event prior to loading the RequireJs modules (life-cycle start) and after the application has loaded these modules (life-cycle complete). The event contains information about which modules have been loaded.

Application Event Type The application can only fire the APPLICATION event from within the application rather than attaching an event raising mechanism from the outside. We choose this approach because we cannot reach the places these events take place reachable from external JavaScript code. To fire this event, we added JavaScript code to the application which utilizes the custom raise functionality of the SRT API. Because Profit Next uses an MVC based architecture, application raises the APPLICATION events whenever a model performs a certain functionality. The events contain information about which model initiated the event and what type of functionality was executed.

⁶<http://requirejs.org/>

Attribute	Description
Url	The url metric defines the location the event took place.
Origin	The origin states which component initiated the event.
Type	The type is recorded to further define the event with more detail e.g. a click dom event, or a stack overflow exception.
ID	The id links the events with a life cycle transition together e.g. link the schedule, start, and end lifecycle transition of the AJAX call together.
Session	The events are linked to a specific session initiated by the user.
User Agent	The user agent contains information about the type of device, browser, and browser version.
User Id	The id which identifies a specific user.

Table 6.2: A description for the measured attributes

Metric	Description
Heapsize	The amount of JavaScript heapsize currently used by the application.
Timestamp	The timestamp states when the event was fired accurate to the millisecond.
Added	The nodes added the DOM.
Modified	The nodes modified in the DOM.
Deleted	The nodes removed from DOM.
Heapsize Limit	The maximal amount of JavaScript heapsize available for the application.
Width	The current width of the browser window in pixels.
Height	The current height of the browser window in pixels.
Max Width	The maximal available width of the browser window.
Max Height	The maximal available height of the browser window.
Blockage	The blockage represents the difference between the expected milliseconds waited and the actual expected milliseconds waited.

Table 6.3: A description of metrics where we obtain values from

6.2.3 Mining Logic and Abstraction Logic

The mining logic defines where and how the implementation measures the acquisition criteria. Because this research focuses on the client, we obtain values for the event types, metrics, and attributes described in the acquisition criteria on the client. We implemented the prior described SRT API to fire the described events. The API obtains values for the metrics and adds the attributes by extending global JavaScript objects with additional JavaScript code. We use the additional code to fire the events, obtain values for the metrics, and add the attributes at the right time.

As we defined in the previous chapter, the abstraction logic describes the logic which we used to analyze the set of events into functions and stages. As we described, functions and stages are subtypes of an activity. To show that functions and stages are not the only subtypes of an activity, we added another subtype named process. We define a process as a set of actions the user performs to achieve a goal. The following subsections describe how the gathered events serve as a base to define the stages, functions, and processes.

Identify Stages Given a trace of events, we filter out the events except for the events referring to the DOM, namely the DOM MANIPULATION events (DM) and the DOM events (DE). Looking at an example event log over time for a single URL depicted in Figure 6.3 we identify three stages. We identify the same stages as described in the previous chapter and use the same logic to analyze them. We use the following statement for the analysis: $\langle DM^* \rangle$, $(\langle DE; DM^* \rangle)^*$; $\langle DE; DM^* \rangle$. Depicted in the figure, the first stage named "page build up" consists out of the first DOM MANIPULATION events prior to the first DOM event ($\langle DM^* \rangle$). During the second stage, named the user interaction stage, the user can execute functions $(\langle DE; DM^* \rangle)^*$. The last stage contains the last DOM event and the following DOM MANIPULATION events on the same URL ($\langle DE; DM^* \rangle$).

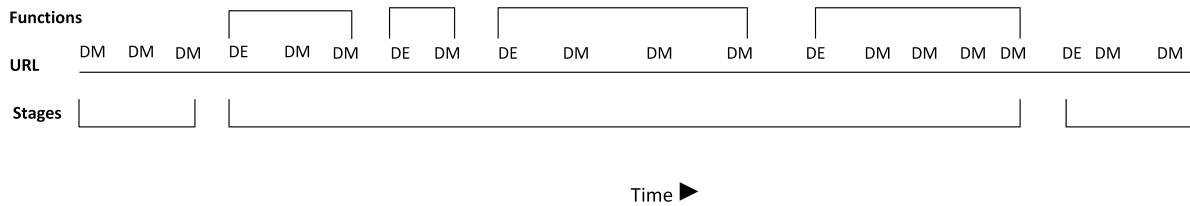


Figure 6.3: Example event timeline occurring on a single URL. The timeline contains DOM Manipulation events (DM) and DOM events (DE)

Page Build Up Looking at Figure 6.3 we can see DOM Manipulation events as the first couple of events the application fires on a URL. After these events we see a DOM event which indicates that the user has started to interact with the application. Therefore we conclude that the DOM Manipulation events leading up to the first DOM event create the requested page so that the user can interact with the application. We call this first stage the page build up stage. During this stage, the application loads the necessary modules and the application modifies the DOM so the user can interact with the application. The page build up starts with the first DOM Manipulation event at a certain page and end at the last DOM Manipulation event before the first DOM event ($\langle DM^* \rangle$).

User Interaction After the first indication of user interaction (the first DOM event) we identify a stage in which the user interacts with the application. We identify the duration of this interaction by looking at the first and last DOM event ($(\langle DE; DM^* \rangle)^*$). Note that when looking at the unfiltered set of events, events can occur between the user interaction stage and the other stages. We assume that these events do not belong to a stage. Because this stage includes both HIT and SRT, this stage cannot be used to improve SRT. Therefore, we include this stage in the analysis but do not use it to draw conclusions from about the architecture.

Page Breakdown After we identified the last interaction of the user on an URL we can conclude that the last interaction of the user resulted in a change in URL. Therefore we specify the events from the last user interaction (DOM event) to the last recorded event on the page as the result of breaking down the page. We call this last stage the page breakdown stage **DE;DM***. We define the page breakdown stage as always initiated by a user action.

Identify Functions We define a function as the period between the user action and the response of the application. In web applications, we translate the user actions as DOM EVENTS and communication of the system's response to DOM MANIPULATION event. Therefore when we look at the filtered trace we define a function as the events between a DOM EVENT and the last DOM MANIPULATION event before the next DOM event (**<DE;DM*>**). Note that the page breakdown together with the page build up of the following URL always result in a function. (Multiple) functions can also occur in the user interaction stage. Therefore, a function does not always span multiple pages.

When we look at the unfiltered list of events we make the following assertions.

- Because a function always starts with a user interaction, we assume that the user cannot perform another action before the prior function finishes. Therefore we assume that functions do not overlap.
- Because we assume functions do not overlap, we also assume that events between the start and end of a function belong to that specific function.
- Because functions are not continuous, we see events occurring between the end of a function and the begin of another function. Therefore we assume that events between the end of a function and the start of another do not belong to a function. We consider these events internal application processes e.g. keep alive requests.
- Functions can overlap multiple pages (e.g. when a user navigates to another page). Therefore we assume that when a function begins on a page and ends on another, the function belongs to the page the function begun on.

Identify Processes Profit Next dedicates every URL to a specific process, therefore the sequential events occurring on a specific URL in a user session belong to that specific process. As an example, when the user navigates to the "create new person" page, the user performs the "create new person" process. Within Profit Next, the user can perform only one process on every URL. Therefore we can identify these processes by measuring the events on a URL. We assume that when the URL in the event log changes, a new process begins and the old process has been completed.

6.2.4 Data Gathering

During the second phase of the method, named the acquisition phase, we implemented the mining logic and abstraction logic in Profit Next as described in Figure 6.2. Because Profit Next is still in development, the operation data was gathered from the employees developing Profit Next whilst they were testing Profit Next. We installed a Profit Next version with the implemented mining logic and abstraction logic in the production environment. The developers at AFAS use the production environment for testing purposes. We gathered the software operation data for a period of 4 days.

Translate End-User Behavior and Extract Software Operation Data During the 4 day period, we collected 27.376 events. We depicted the events gathered for a certain type in Table 6.4. After the acquisition logic was applied to the software operational data, 605 stages, 310 processes, and 291 functions have been abstracted. Appendix E presents fragments of the stage log, function log, and process log.

Verify Correctness of Analyzed Data Looking at the logs in Appendix E, we see stages and functions with a high max duration and standard deviation. For example, the page build up stage of the N_ArtikelView URL has a standard deviation of 109 milliseconds with a minimal duration of 0 milliseconds and a maximal duration of 6.605 milliseconds. As another example, when looking at the function log, we see the Search_N_AdministratieView. The function has a mean duration of 466

milliseconds, a standard deviation of 231 milliseconds, minimal duration of 231 milliseconds and a maximal value of 2915 milliseconds. Note that the processes and user interaction stages also have a high standard deviation. These processes and stages include the users interact with the application. The standard deviation depicts the variation in which the user interacts with the application. Because of this reason we consider a high standard deviation in processes and user interaction stages normal.

In consultation with the architect, a guideline has been defined for the duration of a function, page build up stage, and page breakdown stage. When we measure a duration of more than 5000 milliseconds we consider the measurement as an abnormally high duration. Investigating these measurements points out any improvements in the acquisition criteria, mining logic, and abstraction logic. From the gathered stages and functions, 26 stages and 30 functions have a duration above 5000 milliseconds. The evaluation of these functions and stages resulted in the identification of the following problems in the acquisition criteria.

- Not only the click, but on certain elements, also the mousedown DOM event should be raised.
- The click on a certain link resulted in the focus on a field. The focus event was raised sooner than the click event, resulting in incorrect results. Because we cannot alter the mechanism which resulted in the incorrect results, we removed this click event.
- The F7 function key should also be included as a DOM event as it can lead to a start of a function and the page breakdown stage.
- DOM events considering the breadcrumb are missing and should also be included.
- DOM events considering the sorting of a list of items are missing and should also be included.
- DOM events considering the expanding and collapsing of items in a list are missing.
- Breadcrumb and navigation component are missing an id, resulting in unnamed functions.

Because these problems skewed the analyzed data, we consider the analyzed data incorrect. Therefore we reevaluate the method from the identification phase.

Event	Number of measurements
AJAX	14.685
DOM	1.182
DOMManipulation	3.182
Exception	193
application	4.906
RequireJs	770
EventLoopBlockage	1.923
Acquisition	188
User	131

Table 6.4: Events Gathered per Type

Reevaluating the Identification phase The verification of the analyzed data indicated improvement points in the acquisition criteria and a change in the abstraction logic. We implemented the DOM EVENT to also fire when a mousedown event has occurred on certain DOM elements. The click event which resulted in only a focus on a field was removed from the element. We implemented the DOM EVENT to also fire when the F3, F6, F7, F10, Escape, Alt+F, Alt+N, Delete, or Enter key has been pressed. We have changes the breadcrumb to a single back button where the click event triggers a DOM EVENT. The click resulting in the sorting of a list of items does now also fire a DOM EVENT. The mousedown

event resulting in the expanding and collapsing of an item in a list fires a DOM EVENT. We also added two events named START PAGE and NAVIGATE PAGE. The application fires both events from within the application using the SRT API functionality. For both events, the application measures the *URL, heapsize, timestamp, and session*. These events provide with information about the functionalities performed by the application. Lastly, we added ids to DOM elements to guarantee a correct name for each function.

The abstraction logic has changed for the identification of functions. As indicated, a function starts with a click DOM EVENT by the user. Because a function keys do now also fire a DOM EVENT, we redefined the function’s start event. A function can now also start with a click, F6, F7, Escape, and Alt+N.

6.2.5 Second Data Gathering

After we described the improvements, we updated the mining logic and abstraction logic according to these improvements. These improvements did not lead to any changes in the architecture. Because Profit Next is still in development, only the employees involved in the development of Profit Next use the application. To simulate users, a user simulation component has been implemented and run for 40 hours simulating 20 different users, using 16 different use cases, and running 6 sessions concurrently.

User Simulation Due to the lack of realistic users, a client load generator was implemented. The load generator utilizes a selenium webdriver ⁷ to automate and simulate usage of Profit Next. To simulate a representative user session, 16 representative use cases have been set-up in consultation with the architect. We set up the load generator using a list of 20 users with varying usernames, passwords, input speeds (slow, medium, or fast), and session lengths (short, medium, or long). The user utilizes a username and password to log into the application with. The input speed simulates different types of users, the speed defines the length between two actions of a use case. We specified the length between two actions as a random number of milliseconds between 1000 milliseconds and 10000 milliseconds (slow user), 1000 milliseconds and 5000 milliseconds (medium user), or 1000 milliseconds and 2000 milliseconds (fast user). The session length determines the consecutive executed use cases by the user. A short user executes a random number of use cases between 1 and 3, a medium user executes between 3 and 5 use cases, and a long user executes between 5 and 10 use cases. When the last use case has been executed, the simulation closes the session. When the load generator starts, the generator randomly selects 6 users. These 6 users then concurrently perform the number of assigned use cases. When the use cases have been executed, the load generator closes the session, chooses another random user, and starts a new session.

Figure 6.4 depicts the architecture of the client load generator in and contains a SELENIUM RC SERVER, and the CLIENT LOAD GENERATOR. The SELENIUM RC SERVER communicates with a pre-configured browser e.g Chrome, FireFox, or Internet Explorer. To simulate client actions onto Profit Next. Figure 6.5 depicts the overall architecture with the implemented user simulation component. The following paragraphs further elaborate the sub components of the user simulation.

Selenium RC Server The SELENIUM RC SERVER is a webservice which receives selenium commands from the CLIENT LOAD GENERATOR through simple GET and POST requests. The SELENIUM RC SERVER interprets these requests and sends them to the browser. To operate the browser, the SELENIUM RC SERVER starts the browser process and automatically injects the SELENIUM CORE JavaScript API. The operations from the CLIENT LOAD GENERATOR are thus communicated through GET and POST request, are interpreted by the SELENIUM RC SERVER, and executed onto the browser using the SELENIUM CORE JavaScript API.

⁷<http://docs.seleniumhq.org/>

Client Load Generator The CLIENT LOAD GENERATOR contains the testing logic. The logic includes concurrent browsers, multiple users, multiple user speeds, multiple session lengths, and the logic to read and perform the defined use cases. The CLIENT LOAD GENERATOR reads the use cases (in csv format) and interprets them as a list of commands. The CLIENT LOAD GENERATOR sequentially sends these commands to the SELENIUM RC SERVER until the SELENIUM RC SERVER has performed entire list of commands. After the commands have been performed, the SELENIUM RC SERVER closes the browser process, randomly chooses a new list use cases (depending in the predefined session length), and the SELENIUM RC SERVER starts a new browser process which the SELENIUM RC SERVER uses to execute these use cases onto.

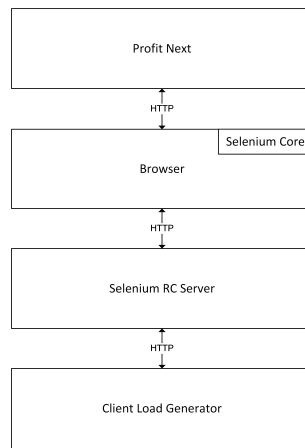


Figure 6.4: Architecture of the Client Load Generator

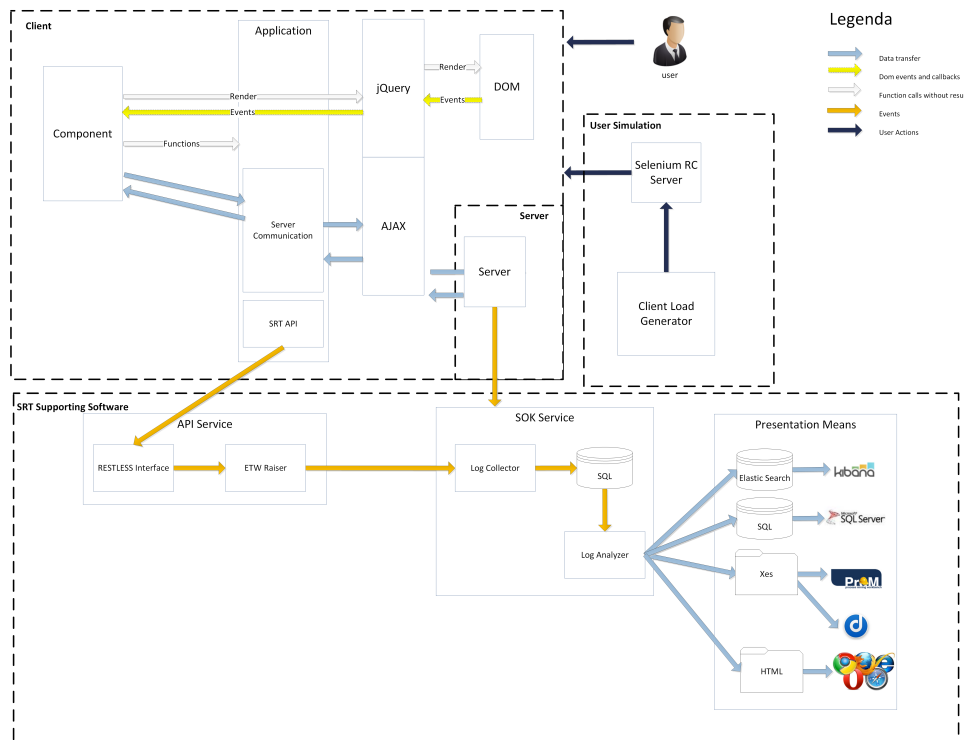


Figure 6.5: Architecture of the Client Load Generator Incorporated in the Overall Architecture

6.3 Data Analysis

Using the client load generator during the period of 40 hours, we collected 7.344.143 events. Table 6.5 depicts the events gathered for a certain type. After the acquisition logic was applied to the software operational data, we abstracted 294.635 stages, 104.294 processes, and 107.188 functions. Appendix F presents fragments of the stage log, function log, and process log.

Event	# Measurements
AJAX	1.223.251
DOM	423.390
DOMManipulation	1.678.124
Exception	54
Application	3.410.286
RequireJs	281.446
EventLoopBlockage	108.165
Acquisition	9.057
User	3.426
Navigate Page	104.2779
Start Page	102.665

Table 6.5: Events Gathered per Type

6.3.1 Verify Correctness of Analyzed Data

By looking at the log of functions, page buildup stages, and page breakdown stages we observe the measured duration either 0 or not 0. Note that because of the user interaction in the processes and user interaction stages, we expect these stages and processes to have a high variation in duration. We consider an abnormally high or low duration always due to the user's interaction speed. Therefore we only use the functions, page buildup stage, and page breakdown stage to measure SRT. By looking at the fuzzy models of a randomly chosen stage depicted in Figure 6.7 we can see a distinction between 0 duration cases and non 0 duration cases. Therefore we decide to split the function and stage logs in a 0 duration log and non 0 duration log.

Looking at the standard deviation of functions, page build up stages, and page breakdown stages we see high standard deviations and maximal durations. For instance, the mean duration of the page build up stage of the N_Administratie_New URL is 10 milliseconds. We measure the standard deviation 85 milliseconds and the maximal duration 3.178 milliseconds. For the C_Persoon.Qmo_Nieuwe_Persoon.Add function, we measure a mean duration of 94 milliseconds. We measure the standard deviation 363 milliseconds and a maximal value of 9842 milliseconds. Because we did not expect these durations and standard deviations, we researched the reason behind these durations.

Looking at the scatter plots depicted in Figure 6.8 outliers (shown within the black elliptic circle) have been identified. To identify the reason for the unexpected duration, each of these functions and stages have been further researched by looking at the event log. As a result, we identified missing events for functions and stages. Looking at the traces of other cases of these outliers indicated that for the outliers, events were missing resulting in the abnormal high duration. However, these events are not missing at the cases with an expected duration. Therefore we conclude that the acquisition logic is correct and that events have been dropped or that the events were not raised. As we described, we expect the ETW mechanism to drop events when with a full event queue.

Because the architect cannot improve a function or stage with an average duration of 0 milliseconds, these functions and stages have been excluded for the rest of the analysis. To identify and remove the outliers, the interquartile range has been calculated for each function and stage. A function or stage with

a duration higher or lower than 1,5 times the interquartile range has been identified as an outlier and has therefore been excluded in further analysis. As an addition, in consult with the architect we decided to consider any function or stage with 10 or fewer measurements as an insufficient number for a useful analysis and. Therefore we omit these measurements. Table 6.6 depicts the result of the exclusion.

	Functions	Stages
<i># measurements</i>	109.903	145.161
<i># 0 duration measurements</i>	12.870	20.662
<i># outliers</i>	4.473	8116
# used in analysis	92.560	116.383

Table 6.6: # Functions and stages excluded

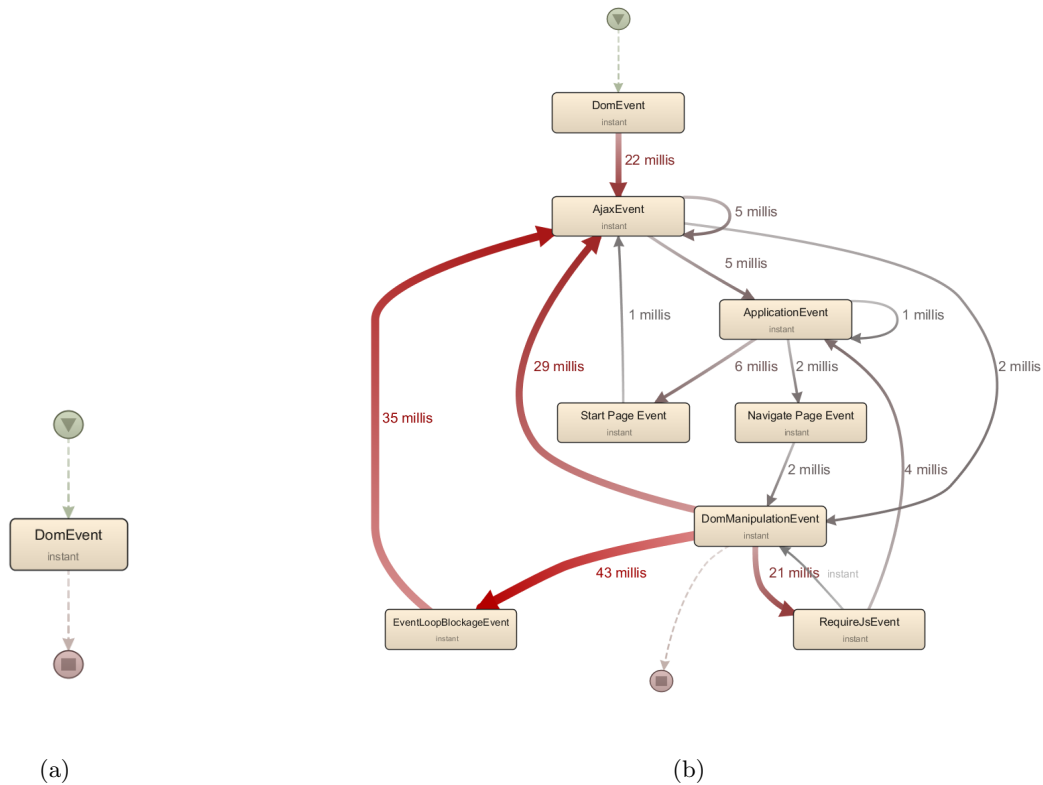
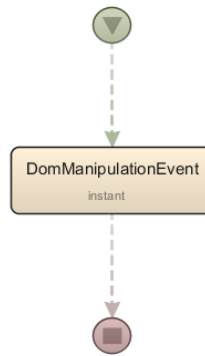


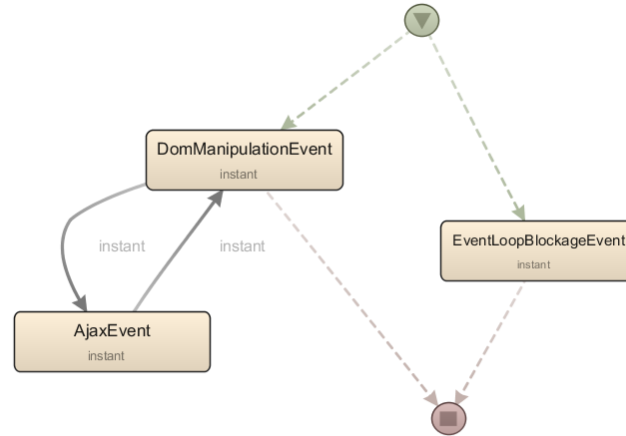
Figure 6.6: Fuzzy Model of the "C_Klant_Qmo_Nieuwe_Klant.Save" function of the cases with a duration of 0 milliseconds (a) and with a not 0 duration (b)

6.3.2 Presentation

After the acquisition phase of the method, we obtained and verified the set of data. During the presentation phase of the method, we describe the presentation means and use the presentation means to visualize the data. These visualizations serve as a base for gaining knowledge and drawing conclusions in the utilization phase.

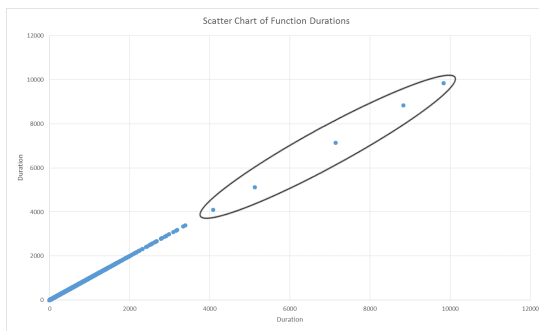


(a)

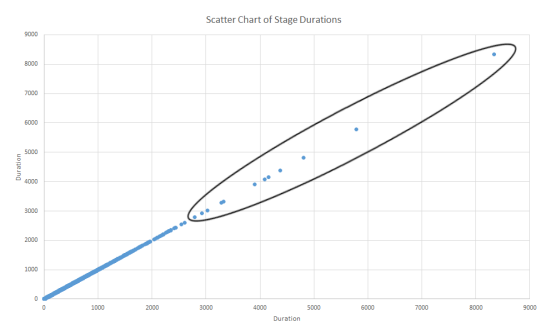


(b)

Figure 6.7: Fuzzy Model of the "Begin" stage on the "N_VoorraadartikelView" url of the cases with a duration of 0 milliseconds (a) and with a not 0 duration (b)



(a)



(b)

Figure 6.8: Scatter plot of functions (a) and stages (b) with the durations plotted against the duration

Define Means of Presentation The following paragraphs provide more information about which presentation means the architect can use to visualize the processes, functions, and stages. Next to a description of the presentation means, we also stated a description on how we utilize the visualizations.

Process To visualize the processes, we created a fuzzy model. For the fuzzy model we created a log with traces consisting of the sequentially executed processes in a user session. The fuzzy model shows the main paths users took in the application. The fuzzy model should depict the 16 use cases we set out to perform. When the data was generated by actual users, the fuzzy model shows the main paths taken by the users. Because users navigate the application differently, the log contains variability in the traces. Therefore we choose to use the fuzzy modeling technique. When the architect knows the main paths of the application, they can optimize the software for these paths. We have depicted the fuzzy model in Figure 6.9 using the session as a case and the processes for each session as events. The model depicts which processes are usually executed after each other.

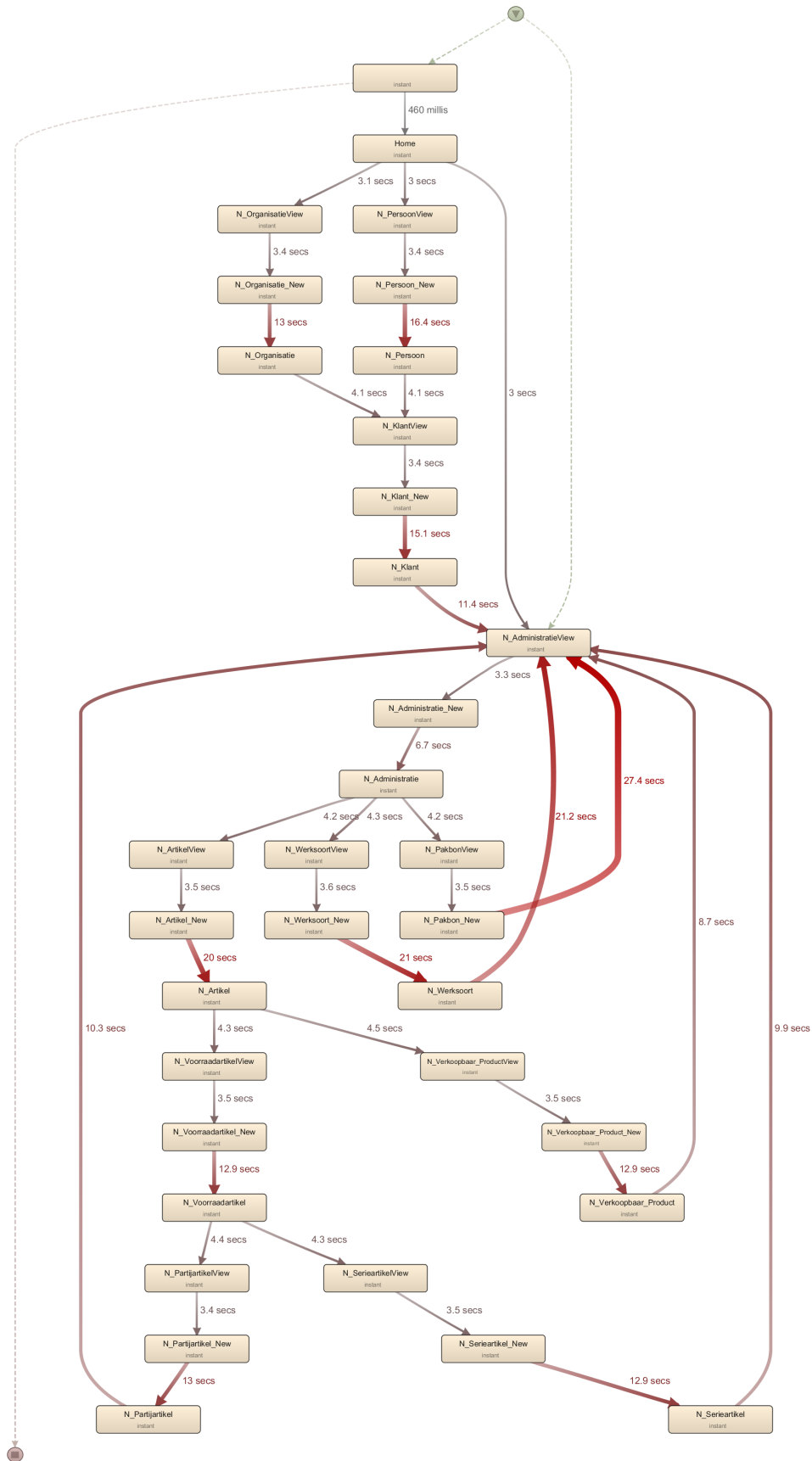


Figure 6.9: Fuzzy model of process execution sequence

Function and Stage A box-plot serves as a starting point to identify improvement points in the functions and stages. When we see a long box-plot, the user experiences a high variation in duration. The experienced variation should be as minimal as possible. A box-plot with a high duration, can indicate an inefficient function or stage. The high duration can also indicate that the function executes complex logic. Note that to create a correct box-plot we have included the outliers in the dataset, the visualization of the box-plot filters out these outliers. After we have selected functions and stages to improve, we use the fuzzy modeling technique. The fuzzy model uses the function or stage as a trace and the events of the function or stage as events. The duration between these events identify the point in which we can obtain the most improvement. A fuzzy model using the sessions as traces and the functions as event provides more information about the order of function execution and can aid in defining representative use cases. Figures 6.10 and 6.11 depict the box-plot models of the functions and stages.

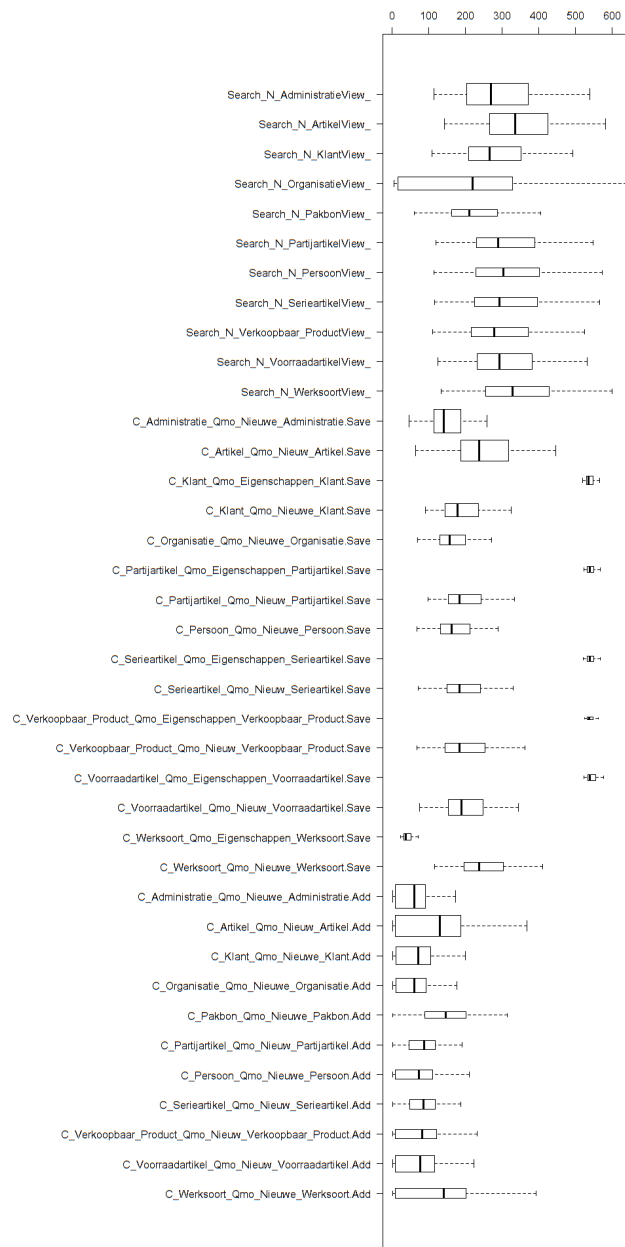


Figure 6.10: Box-plot of the functions with the name on the y axis and the duration on the x axis. The widths of the box-plots are relative to the measurements, more measurements means a thicker box-plot.

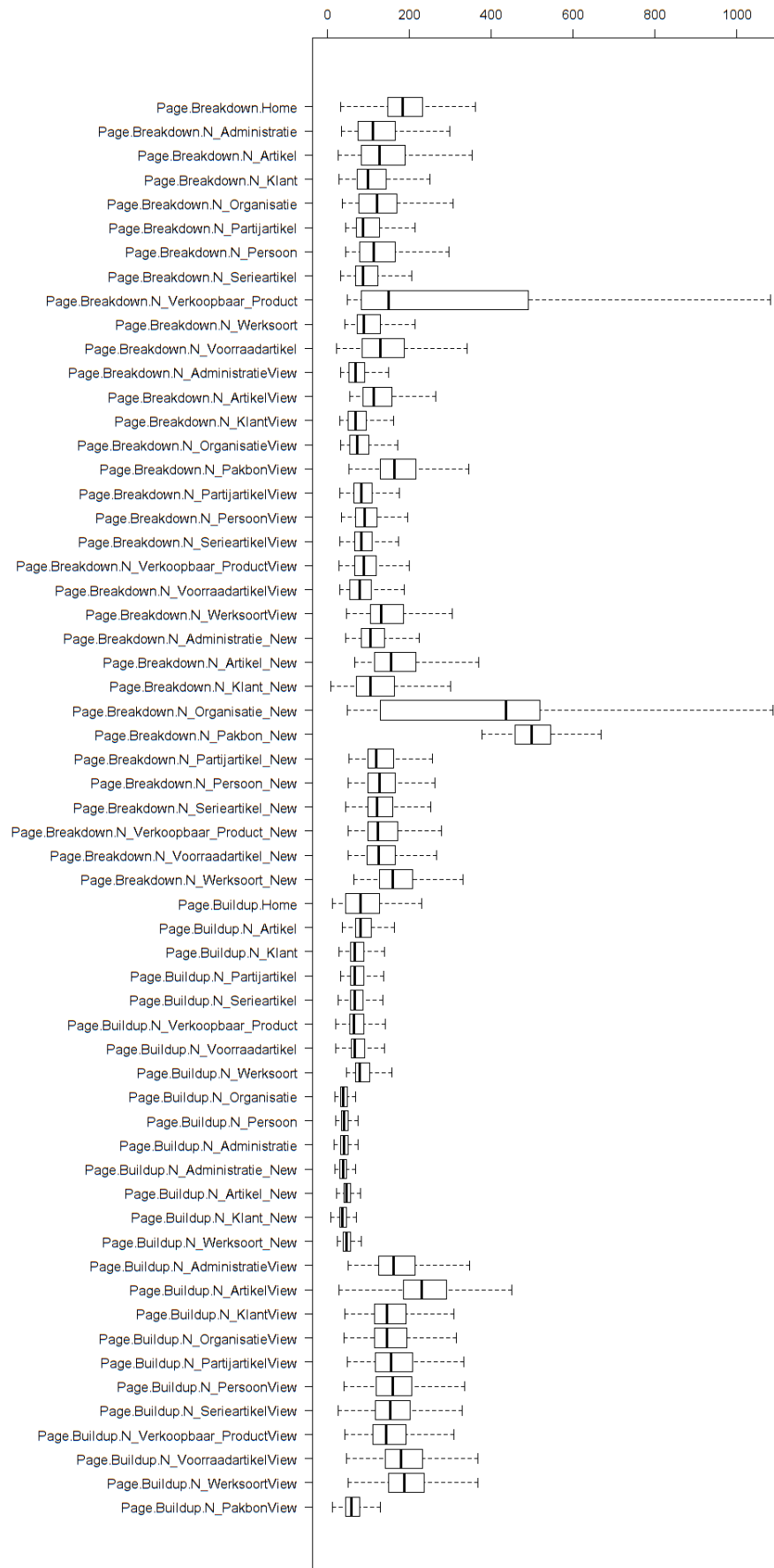


Figure 6.11: Box-plot of the stages with the name and URL on the y axis and the duration on the x axis. The widths of the box-plots are relative to the measurements, more measurements means a thicker box-plot.

6.4 Providing SRT Information

In the previous section we have defined and depicted models which we used to provide the architects with information regarding the SRT of Profit Next. The following paragraphs describe the utilization phase of the method. In this phase we create and use the described presentation means.

Identify SRT Information - Process For the processes we set out to use the process log to create a fuzzy model. The model should indicate the 16 use cases we defined in the user simulation. Looking at Figure 6.12 the 16 use cases can be identified. The figure uses different color lines to depict the use cases. The figure reads from top to bottom. Note that for the readability of the figure, we color coded 6 use cases. In a production scenario, we can use the model to abstract representative use cases. When we would abstract the use cases from an actual production scenario, the architect can improve the application by pre-loading pages that have a high possibility to be opened in the following step. As an example, at the top of the figure, we see that the N_AdministratieView as a central page in the application. In a production scenario, we would advise to pre-load the N_AdministratieView.

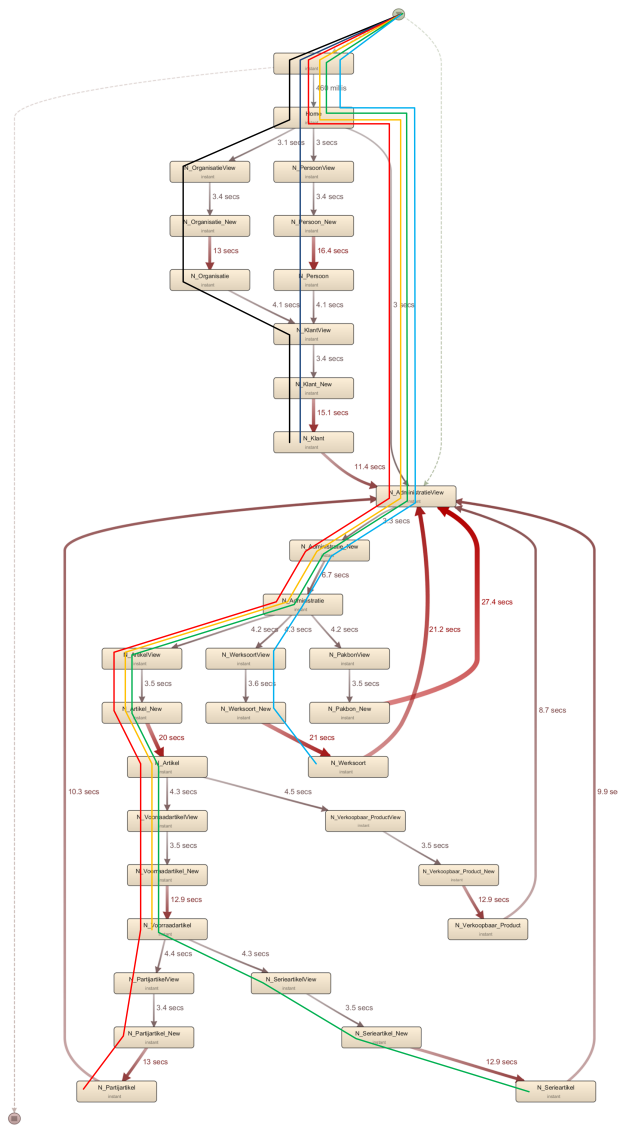


Figure 6.12: Fuzzy model of process execution sequence with use cases indicated with color coded lines

Identify SRT Information - Functions Observations can be made by looking at the box-plot of Figure 6.10. As depicted in Figure 6.13, similar box-plots can be identified. The figure depicts functions with a similar box-plot by a color overlay, e.g. we consider box-plots with a yellow overlay similar. Because of the modular approach of Profit Next, we expect functions with a similar functionality to have similar box-plots e.g. a Search_N_AdministratieView function should have a similar box-plot to a Search_N_KlantView. When we observe a difference in these expected similar functions, the architect should further research the reason for the difference. The difference in box-plot could indicate a difference in execution, which is not described in the architecture. The following paragraphs further elaborate on the observations made for the measured functions.

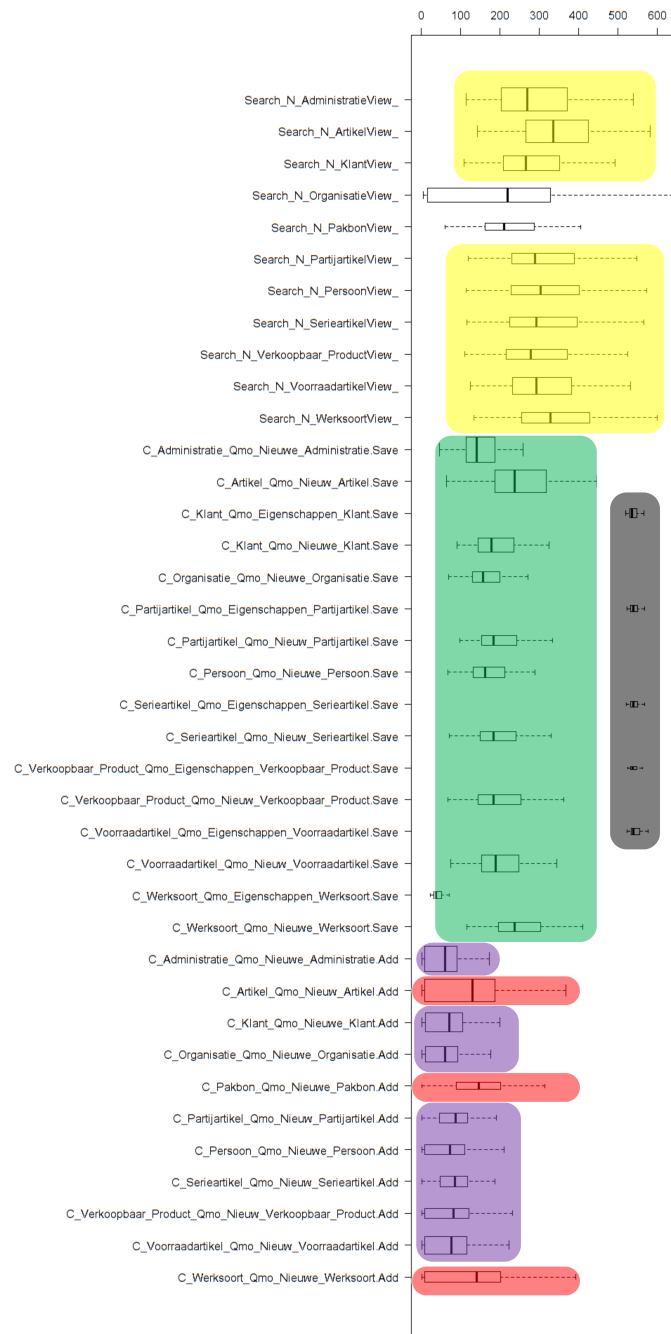


Figure 6.13: Box-plot of the functions with the name on the y axis and the duration in milliseconds on the x axis

Search Functions We identify a search function by its name. When the name contains *Search*, we consider the function a search function. When a user executes a search function, the user requests a page, the application obtains the data needed to show the page from the server, and shows the page. Looking at the box-plots in the yellow overlay in Figure 6.13, the search functions have similar box-plots except for the *Search_N_OrganisatieView* function and the *Search_N_PakbonView* function. Although these functions have different box-plots, we observe that search functions have a high variability in general. Looking at Figure 6.14 we identify the biggest duration located between the AJAX event, DOM Manipulation event and the EventLoopBlockage event. We have measured 6.225 cases. From these cases, the EventLoopBlockage event has occurred in 2.237 cases. In 36.95% of the cases the JavaScript eventloop was blocked with an average of 106 milliseconds. This explains the high variability in duration.

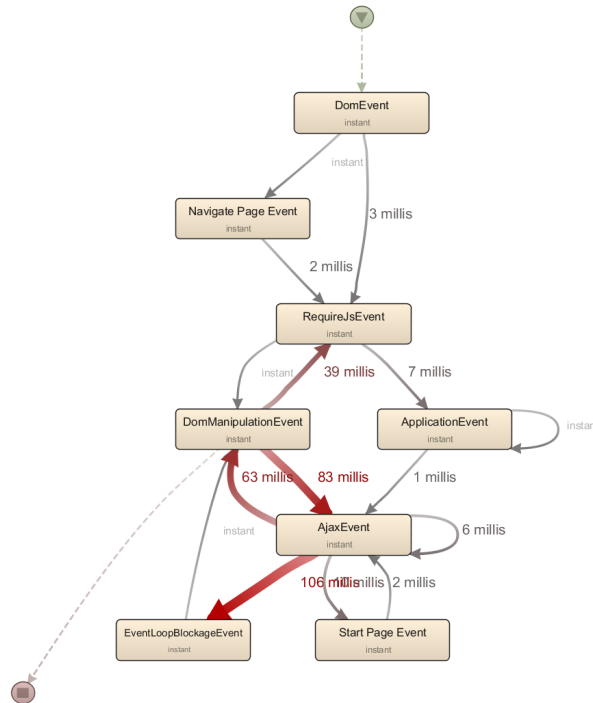


Figure 6.14: A fuzzy model of the "Search_N_ArtikelView" Function, the average duration in milliseconds between events are depicted on the arrows

Edit Item Functions An edit function can be identified by its name. If the name has *Eigenschap-pen* in the name and the name ends with ".Save" we consider the function an edit function. When the user executes an edit item function, the application saves the changes in an existing item. The application then proceeds to remove the edit functionalities and shows the item. As can be seen in Figure 6.13, the edit item functions in the gray overlay have a low variation in duration. These functions have a highly predictable duration. The "*C_Werksoort_Qmo_Eigenschappen_Werksoort.Save*" is the only measured edit function with a different box-plot. The function shares the same low variation in duration but has a lower average duration. Looking at the differences between fuzzy models of the "*C_Werksoort_Qmo_Eigenschappen_Werksoort.Save*" and the "*C_Klant_Qmo_Eigenschappen_Klant.Save*" we observe a difference, see figure 6.15. The "*C_Klant_Qmo_Eigenschappen_Klant.Save*" has an additional link between the DOM Manipulation event and the AJAX event. We observe 497 milliseconds as the average duration between these events. Note that the added 497 milliseconds is about the same difference in duration seen in the box-plot of Figure 6.13.

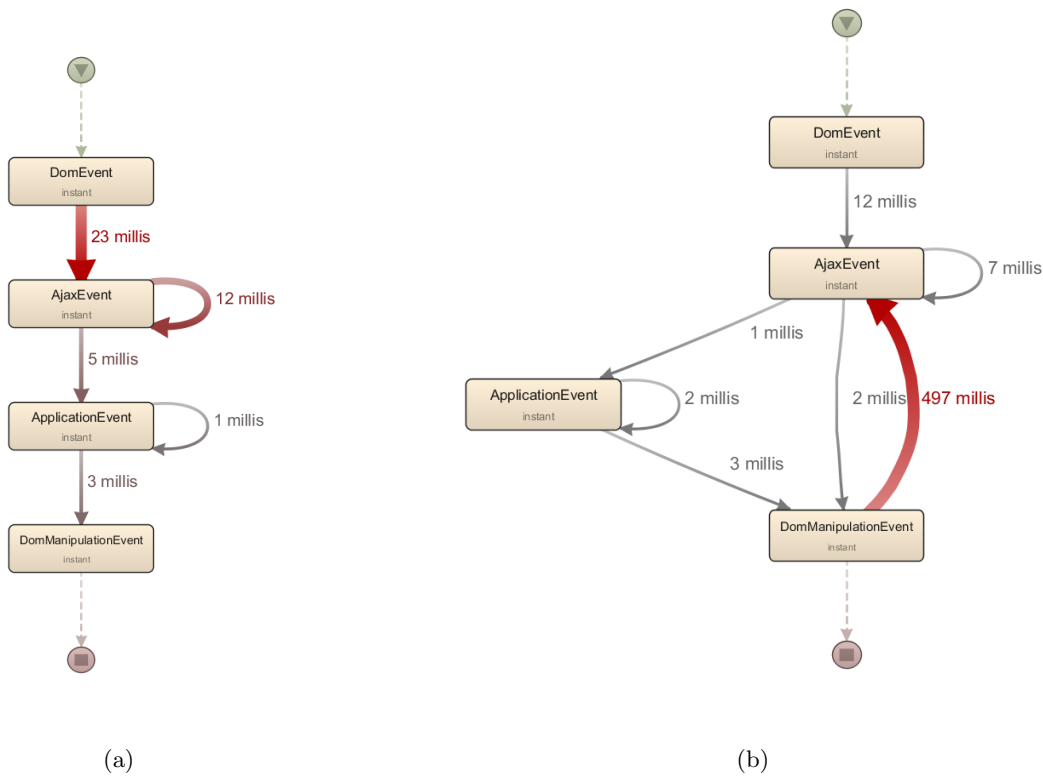


Figure 6.15: Fuzzy model of the "*C_Werksoort_Qmo_Eigenschappen_Werksoort.Save*" function (a) and the "*C_Klant_Qmo_Eigenschappen_Klant.Save*" (b), the average duration in milliseconds between events are depicted on the arrows

Save Item Functions We identify a save item function by its name. When the name contains *Nieuw* or *Nieuwe* and ends with *.Save* we consider the function a save item function. A save function creates a new item and saves it into the database. The application then shows the detail page of the newly created item. In Figure 6.13 the green overlay depicts similar Save Item functions. Looking at this figure, we observe that these functions have a high variability in duration. We do not observe any differences in box-plots for the save item functions. We depicted the fuzzy model of the save item function ("*C_Artikel_Qmo_Nieuw_Artikel.Save*") in Figure 6.16. We can explain the variability in duration by looking at the *EventLoopBlockage* event. This event occurred in 1.012 cases out of the 6.053 cases (16.71 %). That means that the JavaScript eventloop was blocked for 16,71 % of the cases. Blocking the eventloop results in a variation in duration between measurements. The eventloop was blocked with an average of 28 milliseconds (between the DOM Manipulation event and the *EventLoopBlockage* event) and 34 milliseconds (between the *EventLoopBlockage* event and the AJAX event). The eventloop was blocked with an average of 62 milliseconds.

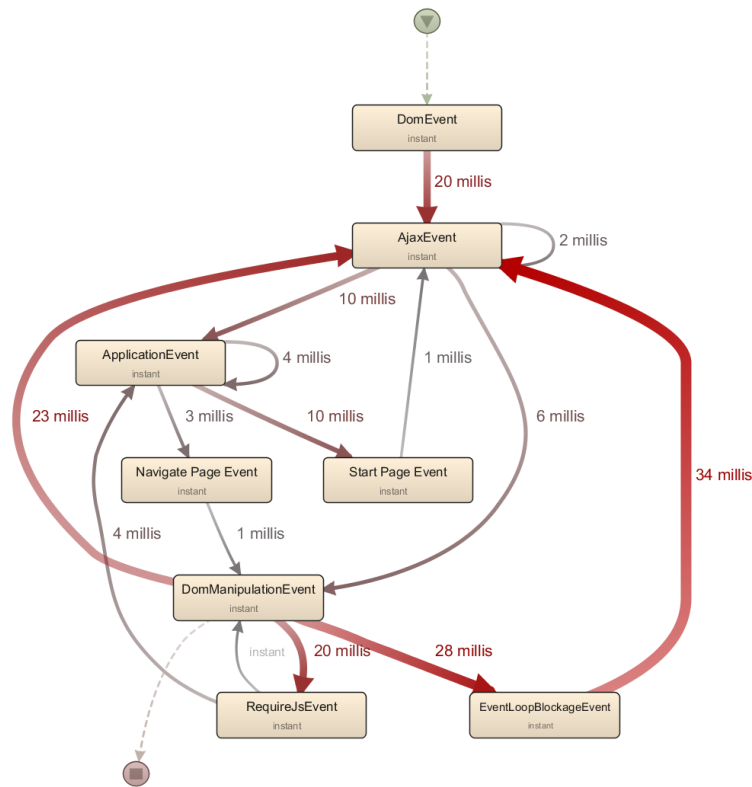


Figure 6.16: Fuzzy model of the "*C_Artikel_Qmo_Nieuw_Artikel.Save*" function, the average duration in milliseconds between events are depicted on the arrows

Add Item Functions We identify an add item function by its name. When the name ends with *.Add* we consider the function an add item function. In an add item function, the application navigates to the page where the user can create a new item. Looking at Figure 6.13 we identify two similar types of add item functions (red and blue overlay). Figure 6.17 depicts the fuzzy models of these two types of Add Item functions. Comparing the events and the structure between the models, we cannot identify a difference.

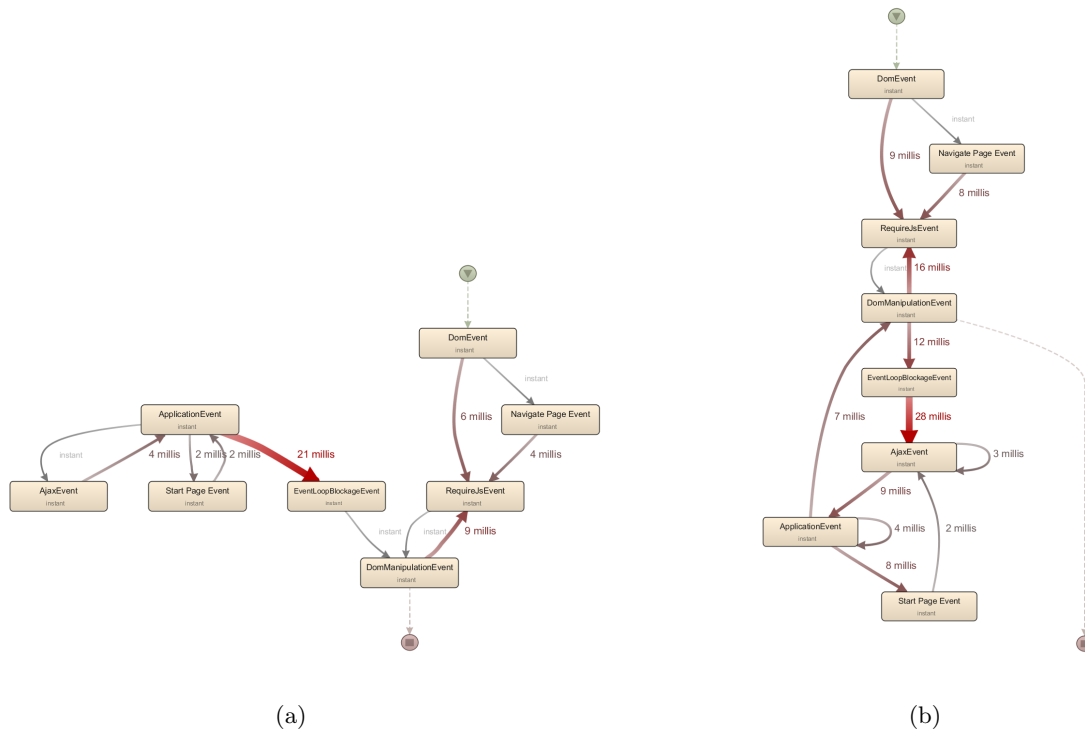


Figure 6.17: Fuzzy model of the "C_Administratie_Qmo_Nieuwe_Administratie.Add" function (a) and the "C_Artikel_Qmo_Nieuw_Artikel.Add" (b), the average duration in milliseconds between events are depicted on the arrows

Identify SRT Information - Stage We can make observations by looking at the box-plot of the measured stages in Figure 6.11. As depicted in Figure 6.18 we identified similar box-plots. As with the functions, we expect similar stages to have similar box-plots. A difference in box-plot could indicate a difference in execution. The architect did not describe the differences in the architecture. Therefore, the architect should research the differences. A color overlay indicates these stages with a similar box-plot, e.g. we consider box-plots with a yellow similar.

In Profit Next, we identified 3 page types. We consider an URL a List View when the URL ends with *View*. A New Item page URL ends with *New*. An Item Detail view when the URL does not end with *View* or *New*. The following paragraphs further elaborate on the observations made for the measured stages.

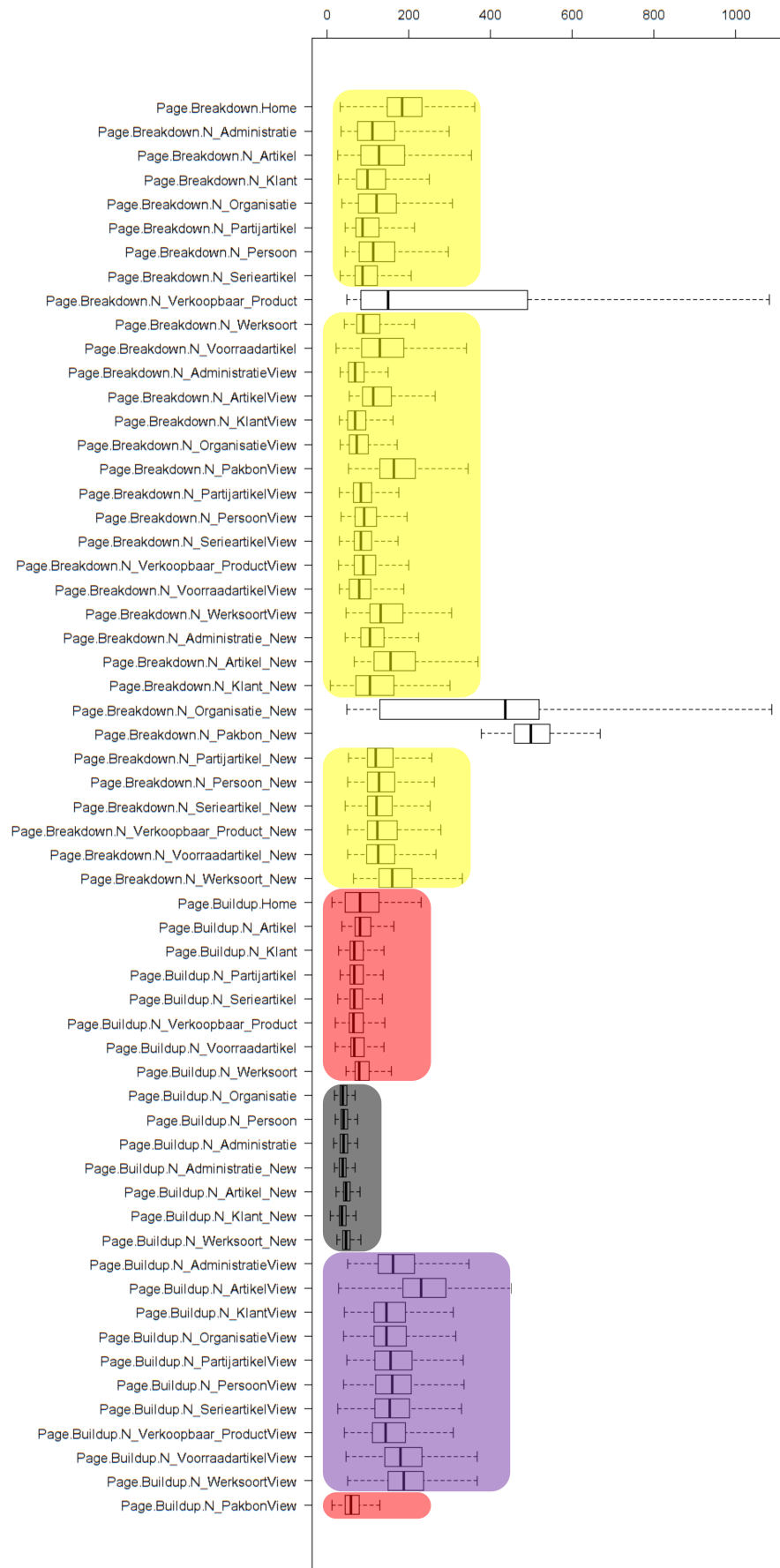


Figure 6.18: Box-plot of the stages with the name on the y axis and the duration in milliseconds on the x axis

Page Breakdown - Detail Page Looking at Figure 6.18, we consider the box-plots of page breakdown stages of detail pages similar except for the Ppage Breakdown stage of the *"N_Verkoopbaar_Product"* View. We explain the difference by comparing the fuzzy models depicted in Figure 6.19. We consider the events and order of events similar. We do observe one difference in the duration between the events. When looking at the duration between the DOM event and the Application event, the *"N_Verkoopbaar_Product"* has an average duration of 473 milliseconds. This order of events leading to the duration occurs in 287 cases out of the 779 cases in total (36,84%).

Looking at the box-plots of the stages, we observe a high variability in duration. To provide the architect with information, we use the Fuzzy model of the *"N_Artikel"*, see Figure 6.19a. Looking at durations between the DOM Manipulation event, EventLoopBlockage event and the RequireJs event we observe an indication for the variation. Out of the total 4.269 cases, the EventLoopBlockage event is fired 729 times (17,07%), delaying the stage with an average of 90 milliseconds after the DOMManipulation event. The RequireJs event is blocked with an average of 72 milliseconds.

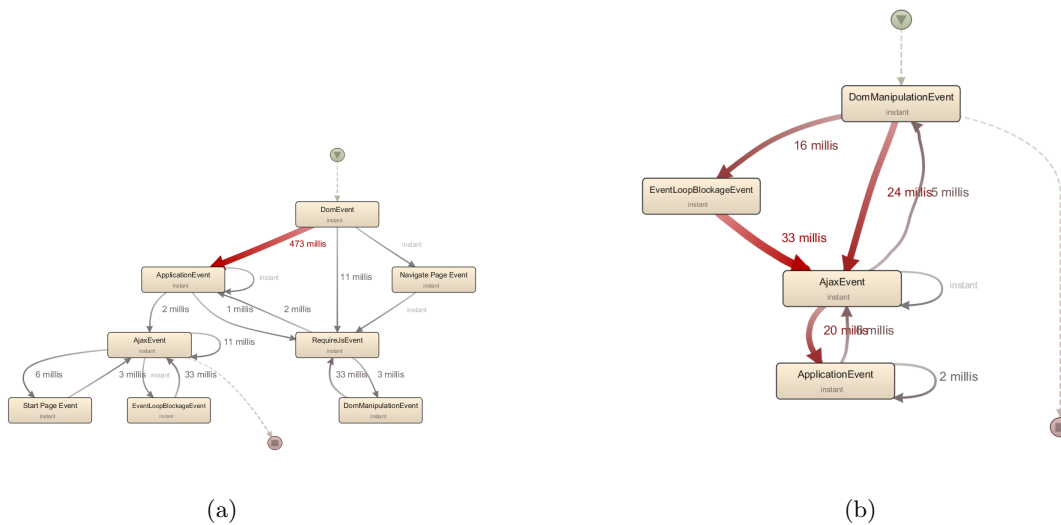


Figure 6.19: Fuzzy model of the page breakdown stage on the *"N_Verkoopbaar_Product"* URL (a) and the same stage on the *"N_Artikel"* URL (b), the average duration in milliseconds between events are depicted on the arrows

Page Breakdown - List Page Figure 6.18 shows that the page breakdown stages of list pages have similar box-plots. Looking at the Fuzzy model of the *"N_PakbonView"*, the duration between the DOMManipulation event and the RequireJs event is disproportionate to the other durations. When improving this stage, the biggest improvement can be made in this duration.

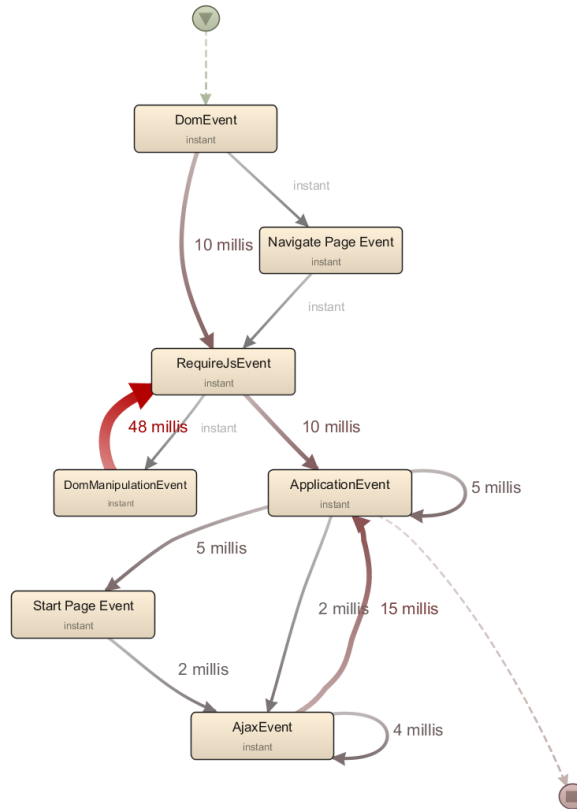


Figure 6.20: Fuzzy model of the page breakdown stage on the *"N_PakbonView"* URL

Page Breakdown - New Page By looking at Figure 6.18, we observe that the page breakdown of the *"N.Organisatie.New"* and *"N.Persoon.New"* have different box-plots than other new page stages. Figure 6.21 gives a comparison between fuzzy models of the *"N.Organisatie.New"* URL and the *"N.Persoon.New"* URL. Comparing the two models, we observe differences. The EventLoopBlockage event in the *"N.Organisatie.New"* takes place in 60 cases out of the 2.444 cases (2,45%). The EventLoopBlockage event explains the variability in the duration to some extent. Another difference in the models is the *"N.Persoon.New"* has an AJAX event directly following the DOM events. The DOM event of the *"N.Organisatie.New"* is directly followed by a RequireJs event.

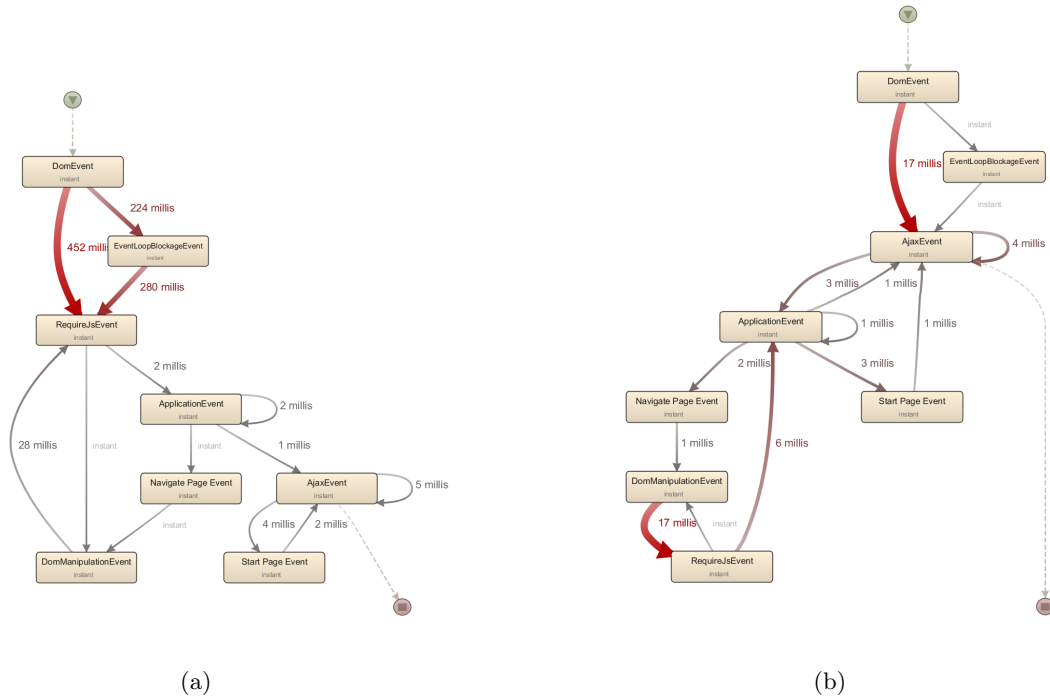


Figure 6.21: Fuzzy model of the page breakdown stage on the *"N.Organisatie.New"* URL (a) and the same stage on the *"N.Persoon.New"* URL (b), the average duration in milliseconds between events are depicted on the arrows

Figure 6.22 gives a comparison between fuzzy models of the *"N_Persoon_New"* and the *"N_Pakbon_New"* stage. Comparing the two models, we observe differences. The EventLoopBlockage in the *"N_Pakbon_New"* takes place in 9 cases out of the 394 cases (2,28%). This explains the variability in the duration to some extend. The *"N_Persoon_New"* has an AJAX event directly following the DOM events. The DOM event of the *"N_Pakbon_New"* is directly followed by a RequiresJs event. Note that the fuzzy models of the *"N_Organisatie_New"* URL and the *"N_Pakbon_New"* URL, are similar. This indicates that when the architect improves one of these two stages, they can improve the other in a similar matter.

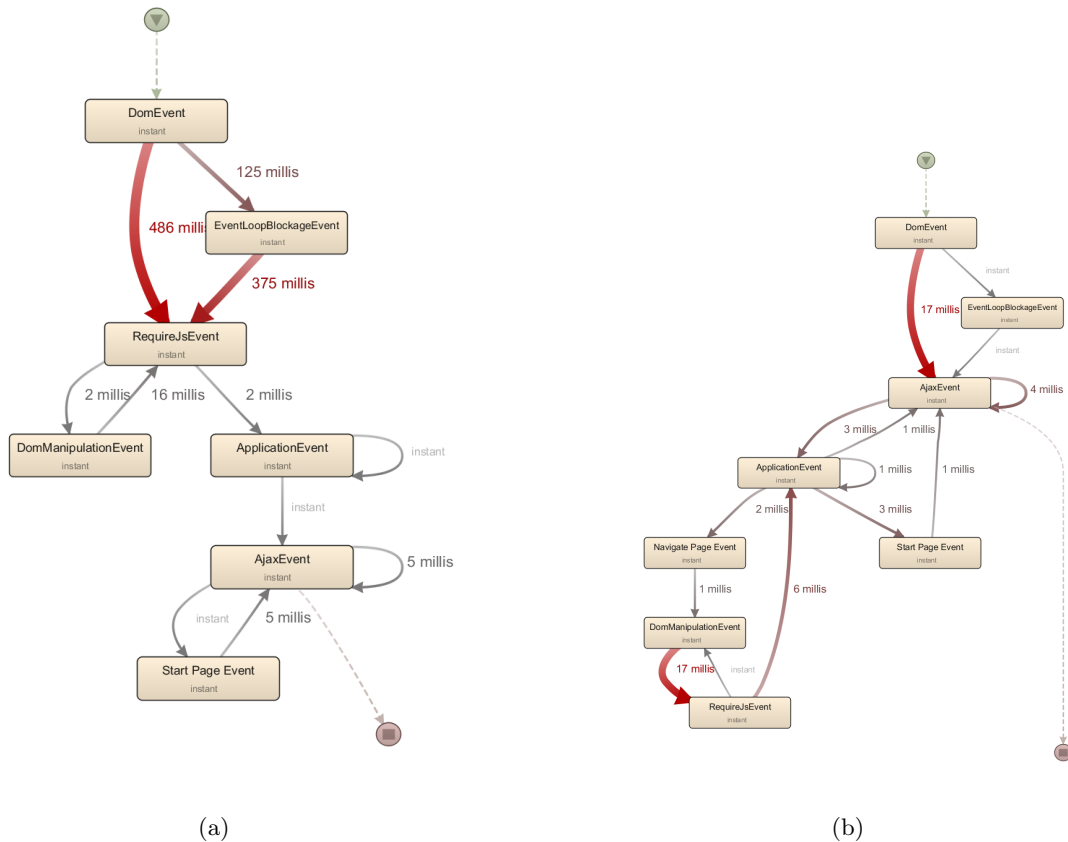


Figure 6.22: Fuzzy model of the page breakdown stage on the *"N_Pakbon_New"* URL (a) and the same stage on the *"N_Persoon_New"* URL (b), the average duration in milliseconds between events are depicted on the arrows

Page Buildup - Detail Page Looking at Figure 6.18, we consider the box-plots of page buildup stages of detail pages similar except for the page buildup stage of the *"N_Organisatie"* and *"N_Persoon"* URL. Figure 6.23 depicts the fuzzy models of the *"N_Artikel"* and *"N_Organisatie"*. Both figures are similar in structure but the model of *"N_Persoon"* shows a higher average duration between events.

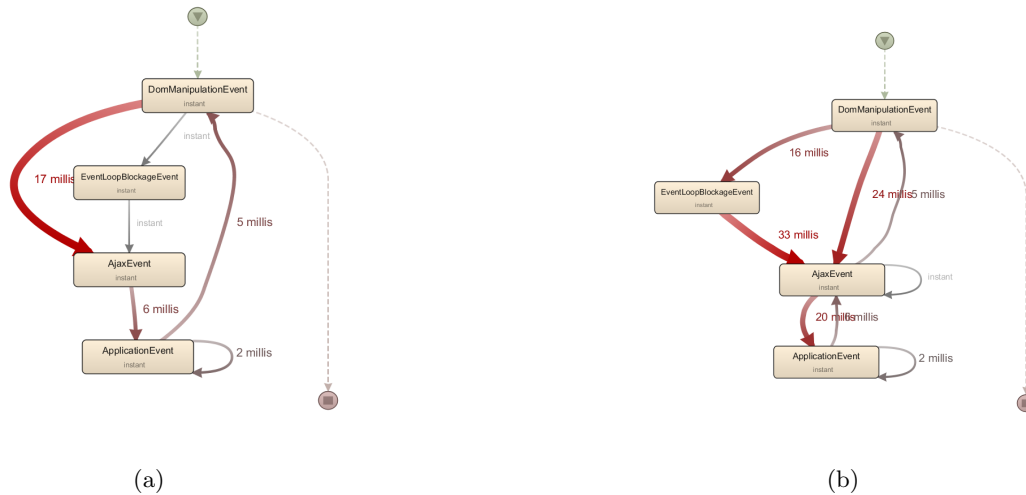


Figure 6.23: Fuzzy model of the page buildup stage on the *"N_Organisatie"* URL (a) and the same stage on the *"N_Artikel"* URL (b), the average duration in milliseconds between events are depicted on the arrows

Page Buildup - List Page By looking at Figure 6.18, we observe that the page buildup of the *"N_PakbonView"* has a different box-plot than the other new page stages. Figure 6.24 depicts the fuzzy models of the *"N_PakbonView"* URL and the *"N_Persoon"* URL. The two fuzzy models look similar when looking at the events. Looking at the percentage of EventLoopBlockage events, the *"N_PersoonView"* (11,67%) shows more than twice the occurrence rate as the *"N_PakbonView"* (5,22%). We also observe in the *"N_PersoonView"* the average duration from the DOMManipulation event to the AJAX event and the other way around, respectively 61 milliseconds and 97 milliseconds. The *"N_PakbonView"* shows a duration of respectively 47 milliseconds and 2 milliseconds. When the architect can decrease the durations of the *"N_PersoonView"* to the *"N_PakbonView"*, we expect that other page buildup stages at list pages can be improved in similar ways.

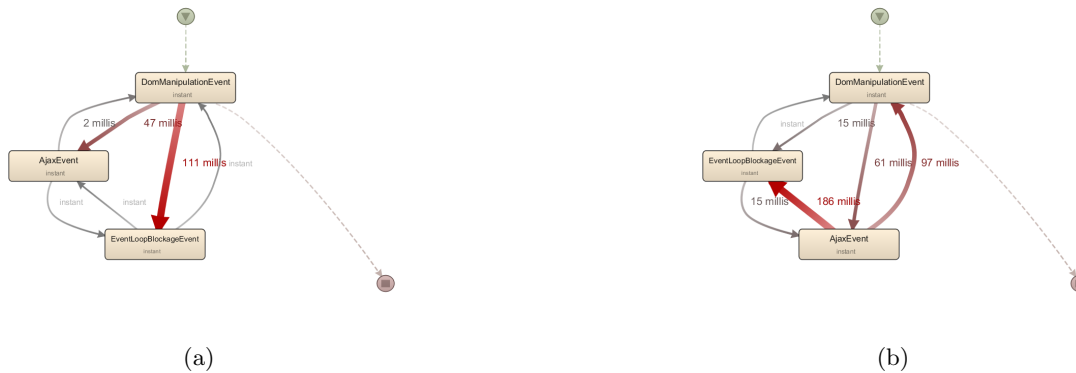


Figure 6.24: Fuzzy model of the page buildup stage on the *"N_PakbonView"* URL (a) and the same stage on the *"N_Persoon"* URL (b), the average duration in milliseconds between events are depicted on the arrows

Page Buildup - New Page Figure 6.18 shows that page Buildup stages of list pages have similar box-plots. Looking at the fuzzy model depicted in Figure 6.25, we observe 3 events with a low duration between them.

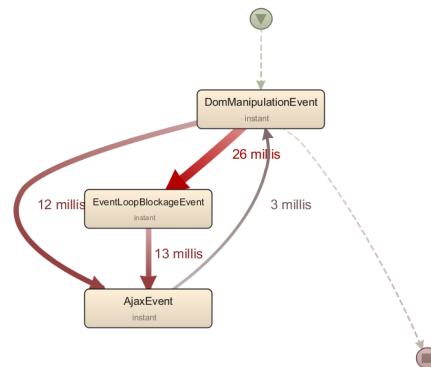


Figure 6.25: Fuzzy model of the page buildup stage on the "N_Administratie_New" URL, the average duration in milliseconds between events are depicted on the arrows

6.5 Future Work

We provided the architects at AFAS with the information stated in the prior section. The following paragraphs elaborate on the usage of the information and provide with further activities needed to improve the architecture. The provided information focuses on a specific level of detail. Although we can make some observations using this level, we need additional levels to draw conclusions from and improve the architecture. For these additional levels of detail, the architect can use the same acquisition criteria as defined in the identification phase. To obtain the additional levels, the architect needs to add to the defined presentation means and visualizations. We advise the architects at AFAS to continue developing and improving the implementation provided by the method. The architects should perform the development and improvement of the implementation according to the method.

A complete solution would be a high level overview which provides information on the measured functions e.g. the average duration, measured eventloop blockages, and errors. When the information on this level indicates a high average duration, a blockage in the eventloop, or errors, the architect can request additional information in further detail levels. The architect constantly alters between the levels of detail to locate the improvement in the architecture.

Functions When the architect observes an EventLoopBlockage event or Error event, they need additional information to draw conclusions and improve the architecture. Looking at the search function of Figure 6.14, we observed high durations between the AJAX event, DOMManipulation event, and the EventLoopBlockage event. In the save item function in Figure 6.16 and the add item function in Figure 6.17 we observed EventLoopBlockage events. We did not observe any Error events in these functions.

Additional levels in detail should provide information about the data size returned by the server. When the application has to process a large dataset, or processes data inefficiently, EventLoopBlockage are more likely to occur. Information about the type of AJAX call can add information about which AJAX call was performed e.g. get a list of items, get a single item, store item. In addition to this information, further levels of detail should provide enough information to answer the following questions:

- Does the EventLoopBlockage event occur only in this function or also in other functions?
- Does the EventLoopBlockage event occur on all devices or only on a specific device?
- Does the EventLoopBlockage event occur on all browsers or only on a specific browser?
- When looking at the stages for this function, does the EventLoopBlockage event occur in a specific stage or in multiple?

Stages When the architect observes an EventLoopBlockage event or Error event in a stage, they need additional levels of detail to draw conclusions from and improve the architecture. Note that for a stage, the same questions arise as defined with the functions. In the page breakdown stage of detail pages (see Figure 6.19) and new pages (see Figures 6.21 and Figures 6.22) we observed EventLoopBlockage events. In the page buildup stage of detail pages (see Figure 6.24), list pages (see Figure 6.24), and new pages (see Figure 6.25) we also observe EventLoopBlockage events. We observe the EventLoopBlockage event occurring after or before a DOM event, Application event, AJAX event, or RequireJs event. Additional levels in details should provide information about the type of DOM event, Application event, AJAX event, or RequireJs event. The architect can use the specific type of event to narrow down the cause of the EventLoopBlockage event.

In the page breakdown stage on the "N_Verkoopbaar_Product" URL (see Figure 6.19), we observe a high duration between the DOM event and the Application event. Additional levels of detail should provide information about the type of Application event. The architect can use this information to further specify the type of Application event that causes the high duration.

In the page breakdown stage on the "N_Organisatie_New" URL (see Figure 6.21) and the page breakdown stage on the "N_Pakbon_New" URL (see Figure 6.22) we observe a high duration between the DOM event and the RequireJs event. Additional levels of detail should provide information about the specific modules loaded resulting in the RequireJs event. The architect can use this information to further specify the loaded module which resulted in the RequireJs event that causes the high duration.

The page breakdown stage of list pages depicted in Figure 6.20 shows no EventLoopBlockage event or error event. We also observe low durations between events. Based on this visualization, the architect does not need any additional levels of detail.

6.6 Method Discussion

In this chapter we utilized the method for improving user-perceived latency (UPL) at the case company AFAS Software B.V. to gain knowledge about SRT in Profit Next. Except for the simulation phase, we executed the method's phases. The method resulted in an increased awareness for the importance of monitoring, measuring, and improving SRT. As a result from the increased awareness, the architects currently improve and add functionality to the implementations we provided during the case study. The architects aim at creating a fully integrated implementation which they can use to automatically monitor, measure, present, and simulate SRT. They aim at automating the process to the extent where the only manual step in the process is to improve the architecture. The following paragraphs provide with a discussion on each phase of the method.

Identification Phase During the identification phase, we described the event types, the application has obtained values for. For each event type, we described the attributes and metrics (the acquisition criteria). These event types were used to define the mining logic and abstraction logic. We used the mining logic and abstraction logic in the next phase to abstract functions, stages and processes from the event log.

Because of the differences in logic for each application, the acquisition criteria varies. Therefore, we could not provide the architects at AFAS with a complete list of acquisition criteria. We relied on the expertise of the architects to define the acquisition criteria. As described in the previous sections, the acquisition criteria was sufficient to provide the architect with information about SRT.

For the abstraction logic, we made assumptions. When we make incorrect assumptions, we compromise the construct validity of the abstraction logic. During the case study we did not observe indications

of an incorrect assumption. However, we did not observe any indications of a correct assumption either. Although we could not validate the assumptions, the architect considered the provided information useful.

Acquisition Phase During the acquisition phase, we implemented the acquisition criteria, mining logic, and acquisition logic. During the implementation we encountered and overcome the obstacles described in the following paragraphs.

Due to technological constraints we could not measure the CPU, RAM, and available bandwidth. To partially cope with these constraints, we measured the eventloop blockage instead of CPU. We measured the JavaScript heapsize and heapsize limit instead of the RAM. Instead of the bandwidth, we measured the device. Note that we do not consider these alternatives equivalent to the CPU, RAM, and bandwidth. However, the metrics can provide with information regarding the CPU, RAM, and bandwidth.

Implementing the JavaScript API to provide with values for the defined metrics was complicated. At first, the API took 500 milliseconds to measure a single event. After optimizations, the user does not notice a difference in performance with or without the API in place.

The implementation of the mining and acquisition logic was, at first, a manually triggered process. We observed a problem with the manual triggering due to the varying amount of events which the logic has to process. Therefore we decided to change the implementation and add an automatic and continuous analysis of the events. This change increased the performance of the implementation and allows for continuous analysis of the gathered events.

Presentation Phase In the presentation phase, we used the analyzed processes, functions, and stages to define and create presentation means. We analyzed the complete set of process logs, function logs, and stage logs to create the XES files used in the presentation means e.g. a fuzzy model. This resulted in running through the entire set of logs multiple times. This approach was sufficient for the obtained logs in the case study. When we apply this approach to a production environment, the size of the obtained logs will increase. Because of this reason, the current implementation does not suffice the needs when used in a production environment. We advise the architects to integrate the presentation means process in the acquisition process. Integrating these processes allow for a continuous analysis and presentation of SRT. The continuous analysis and presentation also allows for a constant monitoring of SRT.

Utilization Phase Based on the models defined and created in the presentation phase, in the utilization phase we used these models to make observations. We provided the architects at AFAS with these observations. Based on these observations, we have provided the architects with recommendations on the subsequent steps needed to improve the architecture.

Simulation Phase Although we provided the architects with information, due to time constraints, we did not perform the simulation action of the method. Therefore we only provided with identifications and did not validate these improvements. The ETW mechanism drops events with a full event queue. Therefore we cannot guarantee that we excluded the results with dropped events by using the 1,5 interquartile range as a selection technique. We created the models by hand. We see a better solution when an implementation can automatically provide with these models.

6.7 Retrospect

Looking back at the case study, we provided the architects of AFAS with a structural approach to monitor, analyze, and improve SRT in Profit Next. The case study resulted in an increased awareness in the importance of monitoring, measuring, and improving SRT. According to the architects of AFAS, we can also use the processes, functions, and stages for other purposes than to monitor, analyze, and improve SRT. The following paragraphs describe other uses for the data we obtained. Note that we have not described a complete set of possibilities. We consider the discovery and research of the possibilities as future work. Although we consider the research on future work, we advise the architects at AFAS to research these possibilities. We reckon these possibilities can lead to added value for not only the architecture but for other departments as well.

We can use the user interaction stage to provide information about the duration a user interacts with a page. The user interface designers can use this duration to measure the effect of interface adjustments. As an example, when they simplify the interface, the average duration should decrease. Using the fuzzy model of the processes, we can provide the user interface designers with information about the sequential processes the user performs. A change in the user interface, can lead to changes in these sequential processes. The designer can also use the model to compare the current model, to the expected model. They can then use the comparison to adjust the interface, either to improve support for the current process model, or to change to interface so that the expected model can be obtained. This can improve the user experience from a user interface perspective.

The architects can use the information about which device type (computer, tablet, or mobile) when migrating the architecture to different devices. They can use the information to substantiate the decision on which functions migrate to which platform. As an example, they see that the user never uses a mobile device to execute a function. They can use this information to substantiate the decision to exclude the function when migrating the application to the mobile platform. This can improve the user experience from a user interface perspective.

The architects can use the information about sequential function execution to create a self learning application. The application can observe functions closely following each other. The application can use the observation to ask the user if it should automate these functions. As a result, the application will automatically execute the functions when it detects the observed pattern. The architect can implement the functionality for each user e.g. the application can automate different functionalities for each user. This can improve the user experience from an artificial intelligence perspective.

Profit Next has functional areas in customer relationship management, course management, document management, financial, fiscal, human resources, logistics, payroll, projects, and workflowmanagement. Using the process fuzzy model can provide more insights in the usage spectrum of these functional area's. The architects can see for a specific customer which functional area they use the most and respond to this by, for instance, allocating additional resources to the functional area. The architects can also use the model to see if a user uses all processes of the functional area. When the user does not use (or partially uses) a functional area, AFAS can provide the customer with additional courses on this area.

As a last example, we measure the errors occurring in the application. The architects can implement a system which notifies AFAS when a user experiences errors. Instead of the user calling AFAS for support, AFAS can then anticipate on the notification and call the user providing with support. This can increase the customer support of AFAS.

Discussion and Conclusion

The previous chapters described the reasons and results of shifting business logic typically residing at server side to the client side. The shift makes user-perceived latency (UPL) more dependable on the user's device. Because of the vast amount of configurations in devices, UPL is difficult to predict and simulate. The research has conceptualized UPL and divided UPL into the system response time (SRT) and human interaction time (HIT). Because of the architectural perspective of the research, we focused on measuring SRT and providing handles to improve SRT. Based on the concept of Software Operation Knowledge (SOK) and the SOK framework, we described a framework fit for measuring and predicting SRT. Based on the framework fit for SRT, we described a method. The method aids and provides handles when defining the data needed to assist with improving SRT. The method aids the architect in defining where from and how to obtain this data. Also included in the method is how the architect can analyze the data into information, and how this information can be transformed into knowledge. After the architect has obtained knowledge on SRT, the architect can further put the method into practice to simulate SRT, predict SRT, and check the SRT prediction. To validate the method, we performed a case study at AFAS Software B.V. where the method was applied on the Profit Next web application to measure SRT. The case study resulted in a definition of data needed to measure SRT, an implementation to obtain the data, an implementation to analyze the data and turn the data into information, and presentation means. Because of the case study, AFAS Software B.V. realized the importance of measuring and improving SRT. The architects at AFAS are currently researching these points provided by the method so that they can improve the architecture. In response to the method, AFAS Software B.V. is building on the implemented architecture and is improving the implementation into an automated solution which will automatically measure, analyze, and identify improvements in SRT.

In the second chapter we defined a main research question and subquestions to answer the main research question. The following paragraphs discuss and answer the sub and main research questions.

SQ1: What data is needed from the usage of web applications to obtain Software Operation Knowledge on user-perceived latency? In Chapter 3, we discussed useful measurements including AJAX and DOM Manipulations. Chapter 4 adds to these measurements with resources, garbage collection, processing, browser, initial application load time. But as described in Chapter 5 the required measurements heavily depend on the type of web application and the opinion of the architect. During the case study, we stored the following data to obtain SOK: RequireJs, the Acquisition API, AJAX, User Interaction (DOM event), DOM Manipulations, Exceptions, Application functions, EventLoop, User, and Navigation. Answering the subquestion, the data needed from the usage of web applications to obtain SOK depends on the type of application and what the architect needs to improve. During the case study, data about DOM events and DOM Manipulations have been essential for the analysis.

SQ2: In what way can data, used to gain Software Operation Knowledge on user-perceived latency be obtained from the users of a web application? In Chapter 5, we advise the architect to implement the architecture as described in this chapter. Within the web application, the architect should implement event raiser components. The architect can implement these components the form of a JavaScript program at the client side or an event raising component on at server side. The latter component depends on the type of server side components. We advise that these components send the gathered data to a single SOK Service which stores the needed data. Answering the sub question, for each component on the client and server the architect should implement a program to raise events. During the case study, a JavaScript API was implemented to obtain the data leading to SOK on UPL. Implementing event raising programs on server side components can provide with a more in depth measurement of UPL.

SQ3: How can Data Mining and Process Mining be used on the data to obtain Software Operation Knowledge on user-perceived latency? To have the ability to perform process mining on the dataset, Chapter 5 advises the architect to store the data in the form of events. The chapter also depicts a model aiding in the defining of these events and their measurements. Using the event format, Chapter 5 defines how the architect can store the events in the XES format (used in process mining tools). Because of the link between XES and the defined events, the architect can use process mining tools supporting this format e.g. Prom Tools and Disco. After the architect has described the events, the architect describes the data mining logic which transforms the obtained events into stages, functions, and possibly more structures. To answer the sub question, when the architect stores the data as events, we can make a link to XES. The link makes it possible to perform process mining on the dataset. The architect can use data mining to transform a set of event into functions, stages, and other structures. The case study provides with an example of another structure, named a process.

SQ4: How can Software Operation Knowledge improve user-perceived latency? Described in Chapter 5 and validated in Chapter 6 Software Operational Knowledge can indicate improvements in UPL. Using fuzzy models (when the log contains variability in the events) or heuristics models (when the log does not contain variability in events), the architect can obtain a model for an entire trace of data structures (stages, functions, or other). By looking at the durations between the events within a trace and taking into account the occurrence rate of these durations, we can identify bottlenecks, we can explain variations in duration, and give an accurate direction to the architect to improve UPL. We can provide other improvements by comparing the model of these data structures. As an example, we expect two data structures (A and B) which perform similar functionalities to result in similar models. When both models show the same structure but B has a bottleneck where A does not, the solution in resolving the bottleneck in B might be found by researching why A does not have a bottleneck. Answering the subquestion, fuzzy or heuristic models can aid the architect when identifying bottlenecks between events, bottlenecks between events in data structures (functions, stages or more). Comparing models can give more direction to improving UPL.

SQ5: What data needs to be obtained from the users of a web application to create a test environment which simulates a measured user-perceived latency situation? As described in Chapter 5, the architect creates the test environment by replicating the application on which the architect has measured SRT. After they measured comparable results on SRT on the test environment, they improve test environment. To obtain comparable SRT measurements on the test environment, the architect uses a representative set of use cases. To answer the subquestion, the data needed to create a test environment which simulates a comparable SRT situation is a representative set of use cases.

SQ6: How can a structured approach describe the creation of a test environment and the obtaining of the necessary data? As described in the previous paragraph, the data needed to create a test environment with comparable SRT measurements is a representative set of use cases. The architect can obtain this set by using data mining logic to analyze the obtained events from the production environment into functions and processes. The architect can then use these functions or processes to create a fuzzy model which they use to base the use cases on. In the case study, we obtained the representative set by using the obtained processes to create a fuzzy model. We used the model to

base representative use cases on. To answer the subquestion, the architect can obtain the data needed to create a test environment by using the data mining logic to obtain a set of functions or processes. The architect can then use the log of functions or processes to create a fuzzy model which they can use to base representative use cases on.

SQ7: Which actions need to be performed to recreate user-perceived latency on a testing environment? As described in Chapter 6, a model using either the functions or processes as events as trace and the user session as cases can show typical user activity in the application and in what order. Based on these typical user sessions, the architect can define representative use cases. When the client load generator executes these use cases as described in Chapter 6 the architect can simulate a representative usage of the application. Answering the subquestion, a fuzzy model of the functions or processes shows the typical usage of the application. Based on the typical usage, the architect defines representative use cases and executed these by using the client load generator. The measurements resulting from this execution recreate SRT on the testing environment.

SQ8: How can the recreated user-perceived latency on a testing environment be used to give a prediction on the actual improvements of user-perceived latency of web applications? At first, the architect builds the testing environment as a replica from the target application. After the architect has obtained comparable results on SRT from the testing environment, they adapt the environment to improve SRT. Because of the comparable results, when architectural changes in the test environment result in SRT changes, we expect the same changes to occur when we apply the same architectural changes in the production environment. Answering the subquestion, because the testing environment is a replica of the production environment, we expect the architectural changes in the testing environment to have the same effect on SRT as on the production environment.

SQ9: How can a method be created to describe the process from Software Operation Knowledge on user-perceived latency to using Software Operation Knowledge for improving user-perceived latency to predict the effect of architectural changes in the software? Based on the SOK framework fit for UPL, we defined and described a method in Chapter 5. To utilize the method, we provided the architect with handles in the same chapter. Through a case study, the method was validated. To answer the subquestion, the method for improving user-perceived latency describes the process utilized to obtain SOK on UPL, and using SOK to improve UPL.

RQ: How can Software Operation Knowledge assist in predicting the effect of changes in the software’s architecture on user-perceived latency of web applications? To obtain in-the-field knowledge of an application we used SOK. Based on the SOK framework, we altered the SOK framework so that the framework applies on UPL. Based on this framework, we created a method existing out of the following five main activities: identification, acquisition, presentation, utilization, and simulation. In the identification activity, the architect identifies the needed metrics, how they obtain the metrics, and what logic they apply on the measurements to analyze it. In the acquisition activity, the architect implements the logic to obtain the defined data and logic to analyze the data. During the presentation activity, the architect defines presentation means to visualize the analyzed data. In the utilization phase, the architect uses these presentations to identify improvements in SRT. During the last activity, the simulation activity, the architect creates an testing environment. Using the obtained data from the users, the architect defines representative use cases and defines means to simulate these use cases. After these use cases simulate comparable SRT measurements in the testing environment, the architect improves the architecture and the software and uses the same simulation means to simulate the use cases. Based on the data obtained from the test, the architect uses the same simulation means to verify the effect of the changes of the architecture on SRT.

7.1 Discussion

During the case study, we used the method to measure, improve, and test SRT of the application Profit Next. Due to time constraints we did not perform and validate the simulation activity of the method. Therefore we cannot ensure that the provided improvements actually improve SRT. The other activities of the method were performed and validated. Due to technological constraints we could not measure every metric we wanted e.g. network usage, RAM usage, and CPU usage.

Because the application was not yet in production, we could not obtain data from actual usage of the application. To cope with this limitation we created a simulation means based on the architect's idea of a representative user. Nevertheless, we created a structured approach which the architects at AFAS Software B.V. use to measure, improve, and simulate SRT. Therefore we conclude that the method works and can be used to measure, improve, and simulate SRT and improve the architecture of a web the application.

During the analysis of the events, we concluded that there were missing events. These missing events were not fired by the application or dropped by the ETW mechanism. These missing events result in an analysis in which the results were incorrect. To cope with these incorrect results, we used the interquartile range to select the outliers. Nevertheless, we cannot know for sure that these are actual outliers. When the interquartile range does not identify an incorrect measurement as a statistical outlier, we include the incorrect measurement in the analysis. Therefore, we cannot consider the analyzed data as 100% correct. To partially cope with the missing events, the architect uses the simulation phase to recreate the measurements.

We can use the gathered data not only on SRT but also to gather other information. For example, we can use the average time the user stays on the page to identify possible user interface bottlenecks. We can use the analyzed functions together with the used device to abstract a list of which functions are mostly used on which device. When making the application available on multiple devices, the architect can use this list to decide which function to port to which device. When we abstract a list of which URLs the users visited and if we know the available URLs, we know what part of the application the user utilizes the most.

7.2 Future Work

A single case study was performed to partially validate the method. The entire method except for the simulation phase was performed. Future research should perform multiple other case studies to further validate the method. To increase the overall quality of this multi-case study research, we recommend usage of the method described by Jansen and Brinkkemper [41].

During the case study, a total of 7.344.143 events have been collected from 6 uses during a period of 40 hours. When the application is being continuously used throughout the day by thousands of users concurrently, the gathered events quickly becomes an immensely large dataset. As an addition to the storage space, the increased usage also increases the necessity of a high performance event firing and storing mechanism. When the mechanism cannot handle these events, the mechanism drops events, the analysis misses events, the analysis becomes unreliable. Future research should address and provide with handles to cope with or eliminate this issue.

In this research, we took a client side perspective. Measurements from a client side perspective are only suitable to improve SRT at the client side. Although these measurements can improve SRT at client side, the architect can also improve SRT at server side. A change at client side can also result in a change in SRT at server side. When we do not measure both the client side and server side we can overlook potential problems. Further research can provide with information on how the described method can be used to measure, analyze, and improve SRT at the server side. The method can only provide with complete measurements of SRT when the method takes both client side and server side into account.

When including a server side perspective on SRT, we see a subject which research needs to address. The case can arise that there are differences in the measured time between servers or between server(s) and client. During the case study, only the client was subject of measurement. Therefore we can consider the time indications provided by the client comparable between each other e.g. if event X with time A indicates that it has occurred before event Y with time B, we can conclude that event X occurred before event Y. When there are multiple servers in place with different time indications, the same situation as described before does no longer apply.

In this research, the architect defines the metrics used to measure SRT. We assume that the architect knows the needed metrics. A subject for future research is to provide with a complete list of metrics which the architect can use to measure SRT. The result of the research should provide with handles for the architect to select a set of the metrics suitable for the application at hand.

During the case study, we created the presentation means by hand. Future research focusing on defining and creating an automated solution can help the architect in a continuously monitoring state. Online reports where the architects can see the current state of SRT in the application. The architect can perform automatic simulation tests whenever they implemented an architectural change. Including the already automated data gathering and analysis, when the presentation means and simulations are completely automated we can provide the architect with an automatic solution that will continuously monitor SRT.

In this research we provided with a structured approach to measure SRT. We used the approach in a case study to measure and improve SRT of the client of Profit Next. When research has provided information on how to include measurements of the server we can create a complete measurement of the SRT process. When we obtain this information, we can provide the architects with information about the percentage of logic performed on the server and on the client. The architect can use this information to substantiate the placement of logic. The most ideal solution for the architect is to have the ability to choose the placement of logic (client or server) on the fly e.g. with hot code swapping. Future research should provide the architect with a structured approach to improve the decision making and shifting of logic on the fly. The research described in this thesis has shed light on a new perspective of user experience, the architectural perspective.

Bibliography

- [1] J. Conallen, "Modeling web application architectures with uml," *Communications of the ACM*, vol. 42, no. 10, pp. 63–70, 1999.
- [2] J. J. Garrett, "Ajax: A New Approach to Web Applications," 2005. [Online]. Available: www.adaptivepath.com/ideas/ajax-new-approach-web-applications
- [3] H. van der Schuur, S. Jansen, and S. Brinkkemper, "A Reference Framework for Utilization of Software Operation Knowledge," in *SEAA'10: Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, 2010, pp. 245–254.
- [4] E. Putrycz, M. Woodside, and X. Wu, "Performance techniques for COTS systems," *Software, IEEE*, pp. 36–44, 2005.
- [5] M. J. Johnson, C.-W. Ho, E. M. Maximilien, and L. Williams, "Incorporating performance testing in test-driven development," *Software, IEEE*, vol. 24, no. 3, pp. 67–73, 2007.
- [6] A. Mani and A. Nagarajan, "Understanding quality of service for Web services." 2002. [Online]. Available: www.ibm.com/developerworks/webservices/library/ws-quality/index.html
- [7] D. F. Pupius, "Rise of the spa," 2013. [Online]. Available: <https://medium.com/tech-talk/fb44da86dc1f>
- [8] A. Perusse, "JavaScript: Rise of the Single Page Application," 2012. [Online]. Available: andreperusse.com/be/post/2012/08/01/JavaScript-Rise-of-the-Single-Page-Application.aspx
- [9] N.a., "An Intro Into Single Page Applications (SPA)," 2013. [Online]. Available: blog.4psa.com/an-intro-into-single-page-applications-spa/
- [10] A. Hevner, S. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, 2004.
- [11] I. ISO, "IEC 25010: 2011: Systems and software engineering—systems and software quality requirements and evaluation (square)—system and software quality models," *International Organization for Standardization*, 2011.
- [12] I. van de Weerd and S. Brinkkemper, "Meta-modeling for situational analysis and design methods," *Handbook of research on modern systems analysis and design technologies and applications*, vol. 35, 2008.
- [13] L. Paulson, "Building rich web applications with Ajax," *Computer*, 2005.
- [14] A. Mesbah and A. Van Deursen, "An architectural style for ajax," in *Software Architecture, 2007. WICSA'07. The Working IEEE/IFIP Conference on*. IEEE, 2007, pp. 9–9.

- [15] G. Nicol, L. Wood, M. Champion, and S. Byrne, “Document object model (DOM) level 3 core specification,” 2001.
- [16] H. van der Schuur, “Process Improvement through Software Operation Knowledge — If the SOK Fits, Wear It!” PhD dissertation, Utrecht University, November 2011.
- [17] A. Newell and S. K. Card, “The prospects for psychological science in human-computer interaction,” *Human-computer interaction*, vol. 1, no. 3, pp. 209–242, 1985.
- [18] H. R. Hartson and P. D. Gray, “Temporal aspects of tasks in the user action notation,” *Human-computer interaction*, vol. 7, no. 1, pp. 1–45, 1992.
- [19] D. L. Maulsby, I. H. Witten, K. A. Kittlitz, and V. G. Franceschin, “Inferring graphical procedures: the compleat metarnouse,” *Human-Computer Interaction*, vol. 7, no. 1, pp. 47–89, 1992.
- [20] R. Riedl, “On the biology of technostress: literature review and research agenda,” *ACM SIGMIS Database*, vol. 44, no. 1, pp. 18–55, 2012.
- [21] R. Riedl, A. Javor *et al.*, “Technostress from a neurobiological perspective,” *Business & Information Systems Engineering*, vol. 4, no. 2, pp. 61–69, 2012.
- [22] P. Kortum and S. C. Peres, “The relationship between system effectiveness and subjective usability scores using the system usability scale,” *International Journal of Human-Computer Interaction*, 2014.
- [23] J. M. Carroll, “Human-computer interaction: Psychology as a science of design,” *International Journal of Human-Computer Studies*, vol. 46, no. 4, pp. 501–522, 1997.
- [24] G. Fischer, “User modeling in human-computer interaction,” *User modeling and user-adapted interaction*, vol. 11, no. 1-2, pp. 65–86, 2001.
- [25] D. A. Norman, “Stages and levels in human-machine interaction,” *International Journal of Man-Machine Studies*, vol. 21, no. 4, pp. 365–375, 1984.
- [26] A. Dix, *Human-computer interaction*. Springer, 2009.
- [27] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 2/E*. Pearson Education India, 1998.
- [28] J. Webster and R. Watson, “Analyzing the past to prepare for the future: writing A,” *MIS quarterly*, 2002.
- [29] S. Goldshtein, D. Zurbalev, and I. Flatow, “Web application performance,” in *Pro. NET Performance*. Springer, 2012, pp. 305–333.
- [30] G. Richards, A. Gal, B. Eich, and J. Vitek, “Automated construction of javascript benchmarks,” in *ACM SIGPLAN Notices*, vol. 46. ACM, 2011, pp. 677–694.
- [31] J.-C. Bolot and P. Hoschka, “Performance engineering of the world wide web: Application to dimensioning and cache design,” *Computer Networks and ISDN Systems*, vol. 28, no. 7, pp. 1397–1405, 1996.
- [32] J. Nielson, C. Williamson, and M. Arlitt, “Benchmarking modern web browsers,” in *2nd IEEE Workshop on Hot Topics in Web Systems and Technologies*, 2008.
- [33] D. Flanagan, *JavaScript: the definitive guide*. O’reilly, 2011.
- [34] C. W. Günther and E. Verbeek, “Xes standard definition,” *Fluxicon Process Laboratories*, 2009.
- [35] M. Bramer, *Principles of data mining*. Springer, 2007.
- [36] W. M. Van der Aalst and A. Weijters, “Process mining: a research agenda,” *Computers in industry*, vol. 53, no. 3, pp. 231–244, 2004.

- [37] R. Agrawal, D. Gunopulos, and F. Leymann, *Mining process models from workflow logs*. Springer, 1998.
- [38] C. W. Günther and W. M. Van Der Aalst, “Fuzzy mining–adaptive process simplification based on multi-perspective metrics,” in *Business Process Management*. Springer, 2007, pp. 328–343.
- [39] A. J. Weijters and W. M. van der Aalst, “Rediscovering workflow models from event-based data using little thumb,” *Integrated Computer-Aided Engineering*, vol. 10, no. 2, pp. 151–162, 2003.
- [40] M. Song and W. M. van der Aalst, “Supporting process mining by showing events at a glance,” in *Proceedings of 17th Annual Workshop on Information Technologies and Systems (WITS 2007)*, Montreal, Canada, 2007, pp. 139–145.
- [41] S. Jansen and S. Brinkkemper, “Applied multi-case research in a mixed-method research project: Customer configuration updating improvement,” *Information Systems Research Methods, Epistemology and Applications*, pp. 1–29, 2008.



Project Planning Activities

Main Activity	Sub Activity	Description
Describe Thesis	Write Short Proposal	In order to communicate the rough borders of the thesis the SHORT PROPOSAL is written.
	Write Long Proposal	Based on the SHORT PROPOSAL, the details of the thesis are communicated through the writing of the LONG PROPOSAL.
	Develop Proposal Presentation	Based on the LONG PROPOSAL, the PROPOSAL PRESENTATION allows for the other students and professors to give feedback on the LONG PROPOSAL during a colloquium.
Research	Research Characteristics Web Applications	To aid in the definition of metrics, the architectural characteristics of web applications have to be researched. These WEB APPLICATION CHARACTERISTICS are described in CH3: WEB APPLICATIONS AND THEIR ARCHITECTURE
	Create Framework Based on SOK	As a base for the method, the SOK framework has to be altered for it to be applicable to user-perceived latency. The resulting FRAMEWORK is described in CH4: USER PERCEIVED LATENCY

	Research User Perceived Latency	To measure UPL, we need to put the concept of UPL into perspective. After we have described the perspective, we need to research the elements and resources affecting UPL. After the elements and resources are defined, metrics that measure these elements need to be researched. CH4: USER PERCEIVED LATENCY includes the result of the described research.
Method	Define Identification Phase	The identification phase encompasses a description on how to define the measured events together with their metrics. After we have provided with handles aiding in the definition of the events and metrics, we need to provide with handles for the implementation and analysis of these events. CH5: A METHOD FOR IMPROVING UPL includes a description of the identification phase.
	Define Acquisition Phase	The defining of the acquisition phase includes a description and handles to obtain the Events and perform the defined analysis of these events. CH5: A METHOD FOR IMPROVING UPL includes a description of the acquisition phase.
	Define Presentation Phase	We provide with handles to define and utilize presentation means this phase. CH5: A METHOD FOR IMPROVING UPL includes a description of the presentation phase.
	Define Utilization Phase	In the utilization phase, we provide with handles and descriptions to aid in using the presentation means to base improvements in UPL on. CH5: A METHOD FOR IMPROVING UPL includes a description of the utilization phase.
	Define Simulation Phase	To simulate the effect of architectural changes on UPL, the simulation phase gives a description and handles to set up a simulation environment and how measurements in this environment can serve as base for a prediction on UPL. CH5: A METHOD FOR IMPROVING UPL includes a description of the simulation phase.

Case Study	Create Data Acquisition Layer	During the case study, we create a tool which functions as a DATA ACQUISITION LAYER. We integrate the DATA ACQUISITION LAYER into an existing product. The DATA ACQUISITION LAYER obtains values for the predefined USER PERCEIVED LATENCY METRICS. CH5: A METHOD FOR IMPROVING UPL describes the process of defining and implementing the tool.
	Perform Data Mining	After the DATA ACQUISITION LAYER has gathered and stored data, we have performed data mining techniques onto the dataset. The data mining process results in an EVENT LOG usable for process mining. CH5: A METHOD FOR IMPROVING UPL includes a description of the used data mining techniques.
	Perform Process Mining	After we have performed the data mining activity, we use process mining techniques on the EVENTLOG. The process mining activity aims to abstract PROCESS MODELS from the EVENTLOG. We use these PROCESS MODELS for process discovery, process conformance, and process enhancement. CH5: A METHOD FOR IMPROVING UPL includes a description of the used process mining techniques.
	Define Insights	We abstract SOFTWARE OPERATION KNOWLEDGE by identifying improvement points in a PROCESS MODEL. CH5: A METHOD FOR IMPROVING UPL includes a description how we used the abstracted PROCESS MODELS to create SOFTWARE OPERATION KNOWLEDGE.
Thesis	Finalize Thesis	Using the information gathered throughout the thesis project, we finalize the thesis resulting in the FINAL THESIS. The FINAL THESIS contains answers to the main research question and the sub questions.
	Develop Final Presentation	We communicate the results of the thesis project to the professors and other students by a FINAL PRESENTATION.

Table A.1: A Description per Activity



Project Planning Concepts

Concept	Description
SHORT PROPOSAL	A SHORT PROPOSAL is a document in which the topic and a short description of the thesis is given. The SHORT PROPOSAL is used by the supervisor to approve or disapprove the subject.
LONG PROPOSAL	A LONG PROPOSAL is a document in which the thesis is described in detail. A LONG PROPOSAL contains the objective, problem statement, research questions, research approach, planning, and the scientific and social relevance. The supervisors use the LONG PROPOSAL to approve or disapprove the thesis. When the supervisors approve the LONG PROPOSAL, the supervisors consider thesis feasible.
PROPOSAL PRESENTATION	After the supervisors approve the LONG PROPOSAL, the LONG PROPOSAL needs to be presented during a colloquium. During the colloquium, students and professors help improve the LONG PROPOSAL.
CH2: RESEARCH METHOD	In CH2: RESEARCH METHOD we describe the method of the thesis. The LONG PROPOSAL serves as a base for the second chapter of the FINAL THESIS.
CH3: WEB APPLICATIONS AND THEIR ARCHITECTURE	In CH3: WEB APPLICATIONS AND THEIR ARCHITECTURE, we discuss the following subjects: the SOK FRAMEWORK, results of the literature study on the characteristics related to the architecture of web applications is described.
FRAMEWORK	The FRAMEWORK describes how we adjusted the SOK framework for it to apply to UPL.
CH:4 USER-PERCEIVED LATENCY	In CH:4 USER-PERCEIVED LATENCY we describe the concept of UPL, contextualize the concept, describe the relations between UPL and quality attributes, and describe SOK together with the SOK framework.

CH5: A METHOD FOR IMPROVING UPL	In chapter 5 of the thesis, based on the SOK framework fit for UPL, we describe a method. The method aids in the defining, measuring, presenting, utilizing, and simulating UPL.
DATA ACQUISITION LAYER	We designed and implemented the DATA ACQUISITION LAYER tool which obtains values for, and stores, USER-PERCEIVED LATENCY METRICS.
EVENT LOG	After the DATA ACQUISITION LAYER has gathered data about the USER-PERCEIVED LATENCY METRICS, data mining is performed resulting in an EVENTLOG.
PROCESS MODEL	Based on the EVENTLOG, we extract the PROCESS MODELS. A PROCESS MODEL describes a set of activities, their ordering of execution, and duration of execution triggered by which user and what resources were used during the process.
SOFTWARE OPERATIONAL KNOWLEDGE	From the PROCESS MODELS, we identify improvements in the user-perceived latency. We consider these extracted improvements SOFTWARE OPERATIONAL KNOWLEDGE.
CH6: CASE STUDY REPORT	CH6: CASE STUDY REPORT describes the entire process of the case study. It encompasses how the DATA ACQUISITION LAYER was designed and integrated, how the layer was put into practice to create the EVENTLOG, how the usage of process mining has created the PROCESS MODELS, and how SOFTWARE OPERATIONAL KNOWLEDGE was extracted from the PROCESS MODELS.
FINAL THESIS	The FINAL THESIS is the resulting document incorporating the information gathered during the thesis project. The FINAL THESIS describes the entire process of the project and the results. The FINAL THESIS answers the main research question and the sub questions.
FINAL PRESENTATION	We communicate the FINAL THESIS to the professors and an audience through the FINAL PRESENTATION

Table B.1: A Description per Deliverable



A Method for Improving UPL Concept Description

Table C.1: A Description per Deliverable

Concept	Description
EVENT	An EVENT indicates something has occurred within the application. The type of EVENT indicates what has happened. The EVENTS serve as a base for the MINING LOGIC and ABSTRACTION LOGIC.
EVENT MEASUREMENT	The EVENT MEASUREMENTS indicate which metrics are going to be measured for each EVENT. The EVENT MEASUREMENTS are used to find specific improvements in the architecture.
ACQUISITION CRITERIA	The ACQUISITION CRITERIA describes which EVENTS and EVENT MEASUREMENTS the application obtains values for.
MINING LOGIC	The MINING LOGIC describes how and where the application measures the ACQUISITION CRITERIA.
ABSTRACTION LOGIC	The ABSTRACTION LOGIC describes how the architect can abstract SOFTWARE OPERATION INFORMATION from the SOFTWARE OPERATION DATA.
SOFTWARE OPERATION DATA	When the MINING LOGIC is implemented into the web application, the implemented logic gathers and stores the SOFTWARE OPERATION DATA.
SOFTWARE OPERATION INFORMATION	The architect applies the ABSTRACTION LOGIC onto the SOFTWARE OPERATION DATA. The ABSTRACTION LOGIC, transforms the SOFTWARE OPERATION DATA into a usable format, the SOFTWARE OPERATION INFORMATION.
SIMULATION ENVIRONMENT	The SIMULATION ENVIRONMENT is an environment fit for simulating the end-user behavior so that a similar, or comparable SRT to the production environment can be measured.
SIMULATION MEANS	The SIMULATION MEANS are ways to simulate end-user behavior onto the SIMULATION ENVIRONMENT.

SIMULATED BEHAVIOR	The SIMULATED BEHAVIOR is similar to SOFTWARE OPERATION DATA with the difference that the architect obtains the SIMULATED BEHAVIOR from using the SIMULATION MEANS onto the SIMULATION ENVIRONMENT.
PRESENTATION MEANS	To present the SOFTWARE OPERATION INFORMATION in a way, the SRT IMPROVEMENT(S) can be identified, the correct PRESENTATION MEANS need to be defined. A PRESENTATION MEANS is a means to visualize the SOFTWARE OPERATION INFORMATION.
DATA VISUALIZATION	Using the PRESENTATION MEANS onto the SOFTWARE OPERATION INFORMATION, DATA VISUALIZATION(S) are created.
SRT IMPROVEMENT	An SRT IMPROVEMENT is an identified point in which SRT can be improved.
ARCHITECTURE IMPROVEMENT	Based on the SRT IMPROVEMENT(S), the architect defines ARCHITECTURE IMPROVEMENT(S). These improvements in the software's architecture result in an improved SRT.
SOFTWARE CHANGE	Based on the ARCHITECTURE IMPROVEMENT(S), the architect implements SOFTWARE CHANGE(S). These improvements the software's architecture so that SRT is improved.



Analysis of Example Event Log into Functions and Stages

#	Timestamp	Name	Url	Target
1	1375642943	Dom Manipulation	http://localhost/home	-
2	1375642951	Dom Manipulation	http://localhost/home	-
3	1375642962	Dom Event	http://localhost/home	person edit
4	1375642973	Dom Manipulation	http://localhost/home	-
5	1375642984	Dom Event	http://localhost/home	organization add
6	1375642988	Dom Manipulation	http://localhost/home	-
7	1375642993	Dom Event	http://localhost/home	address lookup
8	1375643033	Dom Manipulation	http://localhost/employee	-
9	1375643042	Dom Manipulation	http://localhost/employee	-
10	1375643052	Dom Event	http://localhost/employee	address lookup
11	1375643063	Dom Event	http://localhost/employee	person edit
12	1375643075	Dom Manipulation	http://localhost/employee	-
13	1375643086	Dom Event	http://localhost/employee	add right
14	1375643091	Dom Manipulation	http://localhost/employee	-

15	1375643092	Dom Manipulation	http://localhost/administrator	-
16	1375643102	Dom Event	http://localhost/administrator	remove rights
17	1375643113	Dom Manipulation	http://localhost/administrator	-
18	1375643125	Dom Event	http://localhost/administrator	add right
19	1375643137	Dom Manipulation	http://localhost/administrator	-
20	1375643141	Dom Event	http://localhost/administrator	person edit
21	1375643152	Dom Manipulation	http://localhost/administrator	-
22	1375643169	Dom Manipulation	http://localhost/administrator	-
23	1375643174	Dom Manipulation	http://localhost/administrator	-
24	1375643182	Dom Manipulation	http://localhost/administrator	-

Table D.1: Partial Example Event Log

Included Event #	Begin	End	Duration (ms)	Name	Url
1,2	1375642943	1375642951	8	Page build up	http://localhost/home
3,4,5,6,7	1375642962	1375642993	31	User interaction	http://localhost/home
7	1375642993	1375642993	0	Page breakdown	http://localhost/home
8,9	1375643033	1375643042	9	Page build up	http://localhost/employee
10, 11, 12,13	1375643052	1375643086	34	User interaction	http://localhost/employee
13,14	1375643086	1375643091	5	Page breakdown	http://localhost/employee
15	1375643092	1375643092	0	Page build up	http://localhost/administrator
16, 17, 18, 19, 20	1375643102	1375643141	39	User interaction	http://localhost/administrator
20, 21, 22, 23, 24	1375643141	1375643182	41	Page breakdown	http://localhost/administrator

Table D.2: Extracted Stage Log from the Event Log

Included Event #	Begin	End	Duration (ms)	Name	Url
3, 4	1375642962	1375642973	11	Person edit	http://localhost/home
5,6	1375642984	1375642988	4	Organization add	http://localhost/home
7,8,9	1375642993	1375643042	49	Address lookup	http://localhost/home
10	1375643052	1375643052	0	Address lookup	http://localhost/employee
11, 12	1375643063	1375643075	12	Person edit	http://localhost/employee
13, 14, 15	1375643086	1375643092	6	Add rights	http://localhost/employee
16, 17	1375643102	1375643113	11	Remove rights	http://localhost/administrator
18, 19	1375643125	1375643137	12	Add rights	http://localhost/administrator
20, 21, 22, 23, 24	1375643141	1375643182	41	person edit	http://localhost/administrator

Table D.3: Extracted Function Log from the Event Log



Log Fragments Resulting from the First Data Gathering

Name	Url	Mean Duration (ms)	Standard Deviation	Minimal Duration (ms)	Maximal Duration (ms)	Number of Occurrences
Page Build Up	N_AdministratieView	176	92,2402799066465	0	1244	6617
User Interaction	N_AdministratieView	3083	2403,37022057538	100	80033	6616
Page Build Up	N_Administratie_New	7	17,0352606470337	0	158	6613
Page Breakdown	N_Administratie_New	125	90,5668590648334	44	1533	6610
User Interaction	N_Administratie_New	6578	3923,64112107034	2281	33127	6610
Page Breakdown	N_Administratie	138	104,180426761949	34	2156	6605
User Interaction	N_Administratie	4004	2236,53104057296	685	18126	6605
Page Build Up	N_Administratie	49	53,1989914880473	0	1805	6605
Page Build Up	N_ArtikelView	243	109,431299088565	0	1654	5028
User Interaction	N_ArtikelView	3181	2345,60199005366	186	66879	5028
Page Build Up	N_Artikel_New	54	66,7712248329392	0	2258	5027

Table E.1: Fragment of the Abstracted Stage Log

Url	Mean Duration (ms)	Standard Deviation	Minimal Duration (ms)	Maximal Duration (ms)	Number of Occurrences
	525	942	57	3882	50
Home	148452	450708	535	2376506	48
N.OrganisatieView	120488	463576	13	2383193	31
N.PersoonView	16356	13599	0	59851	27
N.Organisatie	277698	716418	24	2257779	15
N.Loonschaal	512822	1672310	2993	6075077	13
N.OrderView	423194	1089724	631	3494242	10
N.Persoon	48216	91645	32	275566	9
N.LoonschaalView	179037	314096	853	697554	8
N.Papieren_Order	32800	80354	601	214998	7
N.Stuur_TaartView	85753	173764	1566	439367	6
N.ArtikelView	7239	9359	564	22332	5
N.Btw_CodeView	14196	5797	6785	19696	4
N.Budgetjaar	6733357	13453224	6098	26913194	4
N.Digitale_OrderView	500	414	202	1090	4
N.Budgetverdeling	174171	340837	1499	685406	4
N.Organisatie.PersoonView	1410743	2802154	369	5613932	4
N.BudgetjaarView	37852	20641	14033	50514	3
N.AdresView	296607	472290	2038	841359	3
N.Adres	139	22	122	164	3

Table E.2: Fragment of the Abstracted Process Log

Name	Url	Mean Duration (ms)	Standard Deviation	Minimal Duration (ms)	Maximal Duration (ms)	Number of Occurrences
C_Administratie_Qmo_Nieuwe_Administratie.Add	N_AdministratieView	66	71,7570942426974	0	1674	8673
C_Administratie_Qmo_Nieuwe_Administratie.Save	N_Administratie_New	170	118,383670799633	0	2530	8666
Search_N_ArtikelView_	N_Administratie	370	168,191741697638	4	2566	6611
C_Artikel_Qmo_Nieuw_Artikel.Add	N_ArtikelView	132	132,856771525517	2	3156	6609
C_Artikel_Qmo_Nieuw_Artikel.Save	N_Artikel_New	281	169,78863051539	0	2673	6462
C_Klant_Qmo_Nieuwe_Klant.Add	N_KlantView	79	76,2649133620289	0	1246	4633
C_Voorraadartikel_Qmo_Nieuw_Voorraadartikel.Add	N_VoorraadartikelView	85	81,3767384090957	0	1638	4016
Search_N_VoorraadartikelView_	N_Artikel	331	163,911661640922	125	2525	4016
C_Voorraadartikel_Qmo_Nieuw_Voorraadartikel.Save	N_Voorraadartikel_New	223	146,495446229255	0	2665	3873
C_Organisatie_Qmo_Nieuwe_Organisatie.Add	N_OrganisatieView	70	78,0584246918757	2	1301	3235
C_Klant_Qmo_Nieuwe_Klant.Save	N_Klant_New	171	139,706535026017	0	1520	3172
Search_N_KlantView_	N_Organisatie_New	0	0	0	0	1851
Search_N_AdministratieView_	N_Klant	294	167,389040055045	0	2657	1738
Search_N_AdministratieView_	Home	466	231,904220562576	231	2915	1673

Table E.3: Fragment of the Abstracted Functions Log

Logs Fragments Resulting from the Second Data Gathering

Name	Url	Mean Duration (ms)	Standard Deviation	Minimal Duration (ms)	Maximal Duration (ms)	Number of Occurrences
Page Build Up	N_Administratie_New	10	85	0	3178	8706
Page Build Up	N_Administratie	45	49	0	1805	8561
Page Build Up	N_AdministratieView	167	92	0	1244	8512
User Interaction	N_Administratie_New	6297	3900	3	33127	8503
User Interaction	N_AdministratieView	3004	2248	1	80033	8502
Page Breakdown	N_Administratie_New	118	84	44	1533	8501
Page Breakdown	N_Administratie	130	99	34	2156	8493
User Interaction	N_Administratie	3886	2220	0	13189	8493
Page Build Up	N_Artikel_New	253	1403	0	21160	7207
Page Build Up	N_Artikel	153	961	0	31118	6536
Page Build Up	N_ArtikelView	231	107	0	1654	6493
User Interaction	N_ArtikelView	3093	2150	1	17286	6486

Table F.1: Fragment of the Abstracted Stage Log

Url	Mean Duration (ms)	Standard Deviation	Minimal Duration (ms)	Maximal Duration (ms)	Number of Occurrences
N_AdministratieView	3233	2252	12	80274	8507
N_Administratie_New	6432	3900	53	33267	8503
N_Administratie	4063	2226	24	13638	8498
N_Artikel_New	17871	10273	55	52990	6489
N_ArtikelView	3421	2155	50	17696	6489
N_Artikel	6555	9609	23	85896	6476
N_KlantView	3273	2401	47	72256	4509
N_Klant_New	18965	12499	4	134683	4508
	294	294	3	4178	4441
N_VoorraadartikelView	3333	2132	55	21583	3936
N_Voorraadartikel_New	11891	7060	68	37631	3934
N_Voorraadartikel	5975	5700	24	45330	3885
N_OrganisatieView	3231	2083	40	11671	3151
N_Organisatie_New	12458	6808	30	39066	3147
Home	2409	4143	2	55134	2747
N_Klant	11085	11114	38	59661	2703
N_PersoonView	3301	2168	63	11274	1365
N_Persoon_New	15190	9337	59	89013	1365
N_Persoon	3885	2232	65	11329	1365
N_Organisatie	3939	2227	46	11649	1348

Table F.2: Fragment of the Abstracted Process Log

Name	Url	Mean Duration (ms)	Standard Deviation	Minimal Duration (ms)	Maximal Duration (ms)	Number of Occurrences
C_Administratie_Qmo_Nieuwe_Administratie.Add	N_AdministratieView	65	71	3	1674	8537
C_Administratie_Qmo_Nieuwe_Administratie.Save	N_Administratie_New	170	118	0	2530	8485
Search_N_ArtikelView_	N_Administratie	352	147	3	2566	6502
C_Artikel_Qmo_Nieuw_Artikel.Add	N_ArtikelView	131	132	2	3156	6487
C_Artikel_Qmo_Nieuw_Artikel.Save	N_Artikel_New	281	169	0	2673	6333
C_Klant_Qmo_Nieuwe_Klant.Add	N_KlantView	79	76	0	1246	4529
Search_N_VoorraadartikelView_	N_Artikel	319	152	4	2525	3929
C_Voorraadartikel_Qmo_Nieuw_Voorraadartikel.Add	N_VoorraadartikelView	84	81	2	1638	3925
C_Voorraadartikel_Qmo_Nieuw_Voorraadartikel.Save	N_Voorraadartikel_New	223	144	0	2665	3793
C_Organisatie_Qmo_Nieuwe_Organisatie.Add	N_OrganisatieView	70	78	2	1301	3165
C_Klant_Qmo_Nieuwe_Klant.Save	N_Klant_New	172	139	0	1520	3079
Search_N_KlantView_	N_Organisatie_New	0	0	0	0	1843
Search_N_AdministratieView_	N_Klant	284	154	0	2657	1698
Search_N_AdministratieView_	Home	387	185	2	2915	1656
C_Persoon_Qmo_Nieuwe_Persoon.Add	N_PersoonView	94	363	2	9842	1377
Search_N_KlantView_	N_Persoon	290	150	4	2415	1368

Table F.3: Fragment of the Abstracted Functions Log