MSc thesis

Master's Programme in Computer Science

On Making Architecturally Significant Decisions in Agile Development: Case Study

Olli Laukkanen

May 27, 2020

Faculty of Science

University of Helsinki

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Olli Laukkanen

Työn nimi — Arbetets titel — Title

On Making Architecturally Significant Decisions in Agile Development: Case Study

Ohjaajat — Handledare — Supervisors

Prof. T. Mikkonen, Ph.D. T. Lehtonen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc thesis	May 27, 2020	48 pages

Tiivistelmä — Referat — Abstract

Päätöksenteko on tärkeä osa ohjelmistokehitystä erityisesti kun kyse on ohjelmistoarkkitehtuurista. Ohjelmistoarkkitehtuuri voidaan jopa ajatella joukkona arkkitehtuurisia päätöksiä. Päätöksenteko vaikuttaa siis hyvin paljon siihen millaiseksi ohjelmistoarkkitehtuuri muodostuu. Tämä tutkielma tutkii arkkitehtuurista päätöksentoa ohjelmistokehitysprojektiin liittyen.

Tämä tutkielma esittää tapaustutkimuksen tulokset, joiden pääasiallisena lähteenä on ollut haastattelut. Tapaustutkimuksen tapaus on yksittäinen tapaus, joka tehtiin alihankitun ohjelmistokehitysprojektin aikana. Päätökseen liittyy ohjelmistokehitystiimi ja useita sidosryhmiä asiakkaan puolelta, mukaanlukien arkkitehtejä.

Ohjelmistokehitystiimi reagoi nopeasti päätöksen tarpeellisuuteen kun tarve nousi esille projektin aikana. Suurin osa päätöksentekoon liittyvästä työstä tehtiin ketterän ohjelmistokehityksen lomassa sujuvasti. Vain lopullinen päätös asiakkaalta tapahtui erillään ohjelmistokehitysprosessista. Tämä viimeinen hyväksyntä annettiin vasta jälkeenpäin, kun päätös oli jo käytännössä päätetty ja implementoitu. Tämä kuvaa hyvin kuinka vaikeaa on yhdistää ohjelmistokehityksen ulkopuolista päätöksentekoa ohjelmistokehitysprosessiin tehokkaasti. Tämän tutkimuksen tapauksessa päätöksenteossa oli työnjako, jossa yksi henkilö tutki ja valmisteli päätöksen. Tämän henkilön työ muodosti suurimman osan tehdystä työstä päätökseen liittyen. Tämän tyyppinen työnjako voisi mahdollisesti olla yleistettävissä päätöksentekoon yleisesti ohjelmistokehityksen piirissä.

ACM Computing Classification System (CCS)
Software and its engineering → Software creation and management → Collaboration in software development
Software and its engineering → Software creation and management → Designing software
Software and its engineering → Software organization and properties → Software system structures → Software architectures

Avainsanat — Nyckelord — Keywords

decision-making, software architecture

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software study track

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Olli Laukkanen

Työn nimi — Arbetets titel — Title

On Making Architecturally Significant Decisions in Agile Development: Case Study

Ohjaajat — Handledare — Supervisors

Prof. T. Mikkonen, Ph.D. T. Lehtonen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc thesis	May 27, 2020	48 pages

Tiivistelmä — Referat — Abstract

Decision-making is an important part of all of software development. This is especially true in the context of software architecture. Software architecture can even be thought of as a set of architectural decisions. Decision-making therefore plays a large part in influencing the architecture of a system. This thesis studies architecturally significant decision-making in the context of a software development project.

This thesis presents the results of a case study where the primary source of data was interviews. The case is a single decision made in the middle of a subcontracted project. It involves the development team and several stakeholders from the client, including architects.

The decision was handled quickly by the development team when an acute need for a decision arose. The work relating to the decision-making was mostly done within the agile development process used by the development team. Only the final approval from the client was done outside the development process. This final approval was given after the decision was already decided in practise and an implementation based on it was built. This illustrates how difficult it is to incorporate outside decision-making into software development. The decision-making also had a division of labour where one person did the researching and preparing of the the decision. This constituted most of the work done relating to the decision. This type of division of labour may perhaps be generalized further into other decision-making elsewhere within software development generally.

ACM Computing Classification System (CCS)

Software and its engineering → Software creation and management → Collaboration in software development

Software and its engineering → Software creation and management → Designing software

Software and its engineering → Software organization and properties → Software system structures → Software architectures

Avainsanat — Nyckelord — Keywords

decision-making, software architecture

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software study track

Acknowledgements

I would like to thank Business Finland and Solita for allowing me to study decision-making within their project. I would like to thank my supervisor Tommi Mikkonen from the University of Helsinki for helping guide the work and find what is important. Timo Lehtonen from Solita also deserves my thanks for having acted as my second advisor and given me invaluable assistance with organizing and setting things up with the project team and other relevant people, as well as providing useful notes and insight. I would also like to thank Janne Rintanen from Solita who provided very useful feedback from a different perspective. Finally I would like to thank my employer National Land Survey of Finland for being flexible.

Helsinki, 27. May 2020
Olli Laukkanen

Contents

1 Introduction

Decision-making in a software development project is a non-trivial activity that has a large impact on the resulting system. It often involves several stakeholders with differing knowledge and views. In decision-making different goals that may be conflicting with one another have to be accounted for. It is not always as simple as prioritizing one goal over another since you can have situations where one solution achieves both goals roughly as well but in another solution one goal is achieved very well and the other very poorly. In this case both solutions seem viable if one goal is just blindly prioritized over another.

Software architecture can be thought of as a description of the architecture or as a set of architectural decisions [21]. When the former definition is used exclusively, knowledge vaporization may occur. When the reasoning for various decisions is not documented, as a result later on it may be unclear why an architecture is the way it is. On the other hand, when architecture is thought of as a set of architectural decisions, decisions are documented and it can be discovered later why the architecture is a specific way. Naturally in practise there usually needs to be explicit descriptions, like diagrams of the architecture, both of the intended and produced architectures, along with the documentation of specific decisions that were made. The architecturally significant decisions should, in this case, be considered an important part of software architecture [21]. If the documentation of the decisions is complete enough, architecture descriptions could theoretically be derived from just the decisions, but in practise documentation of architectural decisions is likely never complete enough and deriving diagrams from decisions can be cumbersome, not to mention documenting every little decision that has any impact on the architecture may be very time-consuming and provide little value.

Regardless of how software architecture is defined, decision-making is a large part of the architectural development of a software system. Whether the architecture emerges without any forethought or it is consciously worked towards, implicit or explicit decision-making is how this architecture arises. Therefore studying why specific architecturally significant decisions are made and how they are made is important in order to better understand decision-making and make the best, most optimal architecture

possible.

This thesis aims to provide further insight into decision-making through a detailed analysis of a specific architecturally significant decision made in a subcontracted project. The project in question is done for Business Finland (client) who provide financing for various business ventures [10]. The project is a part of a larger initiative. It is implemented by a Finnish IT-consultancy company Solita (supplier). The decision that was studied has to do with how information regarding a joint venture is extracted from a joint venture application sent to the system and how this information is stored in document management systems. A joint venture refers to a business venture involving several parties that may include, for instance, businesses and/or universities. This decision was selected with the requirement that it was done explicitly and that people were aware of making it in order to study rational explicit decisions specifically. This is opposed to implicit or "accidental" decision-making where people are not aware of a decision having been made. Another requirement for selecting the decision was that that it was done during the project, as opposed to before the project or at the beginning of the project, in order to study how decision-making is done alongside development. The decision was found based on provided documentation about some projects at the client as well as discussions with people that were aware of the projects there and some of the people working there.

Everyone involved in the decision from both the client and the team, that was implementing the solution, was interviewed to establish how the decision was made and what affected it.

The remainder of the thesis is structured in the following way: Chapter 2 contains background information about software architecture, agile software development processes, and decision-making within software engineering, as well as information about decision-making in general. Chapter 3 describes the methods used for the study and contains a detailed description of the case. Chapter 4 contains the results, Chapter 5 contains the discussion and analysis, and finally Chapter 6 summarizes the conclusions.

2 Background

The theoretical background for this thesis is presented in this Chapter. It is structured as follows. First the software architecture is explained in Section 2.1. Architecturally significant decisions are elaborated on in Section 2.2, and decision-making as well as what affects it is explained in Section 2.3. Finally in Section 2.4 agile software development process and decision-making within it is elaborated on.

2.1 Software architecture

Defining architecture is not as trivial a task is it may first appear. There does exist an ISO-standard for what architecture descriptions are (ISO/IEC/IEEE 42010:2011 [20]) which includes a definition for software architecture itself. This definition describes software architecture as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Many agree with defining software architecture this way [11, 28]. This definition does not, however, conclusively define what constitutes as *fundamental* concepts or properties so it is up to the architect to determine what the *fundamental* concepts and properties are for a given project or system. This naturally leaves a lot of room for interpretation.

Overall design of a software system can be split into two: software architecture and detailed design [15]. Detailed design refers to the aspects of the overall design that do not fall under software architecture. Drawing the line between what is software architecture and what is detailed design can be very difficult. There is not a clear consensus on how to draw that line [15]. It may just be that whatever an architect for a given project considers important and relevant is considered architecture. What is important for one architect may not be important for another. However, to get to a more practical level, some things that have traditionally been considered as architectural include, for instance, the composition of components to make up the system, communication protocols, physical distribution, and scaling [35]. It is worthwhile noting that it is not necessary to know or define the architecture of a system fully for architecture to be useful. Even incomplete information can be used effectively to discuss the system.

Architecture may also include low-level details about a system if they are significant in the scope of the whole system and contribute to the quality of the system.

Regardless of how it is defined exactly, software architecture has a large impact on the quality of a system [23]. Quality in this case refers to the non-functional attributes of a software system that are important for the use of the system. What is important in one system is determined by the stakeholders of the system. These kinds of attributes include things like modifiability, responsiveness, availability and fault-tolerance. It may be argued that architecture and the functionality of a system move mostly independent of one another. A system with a specific functionality can be implemented with varying architectures which results in systems that mainly differ in their quality attributes. Therefore it becomes important to figure out what the users of the system require in terms of non-functional quality.

Recently over the last decade or so the perception of what software architecture is has shifted away from only seeing architecture as the documented architecture in the form of architectural pictures etc. and towards including architecturally significant decisions as an important part of what software architecture is [21, 9]. After all, the implemented architecture is a direct result of the various architecturally significant decisions, whether explicit or not. Architectural Knowledge Vaporization is an important concern that is aimed at combating by treating architecturally significant decisions as an important part of the architecture. When it is not known why a certain architecturally, or otherwise, significant decision was made, it may be difficult to change the system the decision influenced. What if the decision had a good, but non-obvious reason behind it?

In practise there are also different versions of the architecture of the system. Often there is a description of the current architecture as well as a description or descriptions of relevant parts of a future architecture that is the target. Naturally there can also be several target architectures for varying timescales. These kinds of target architectures may also be unnecessary.

2.2 Architecturally significant decisions

Architecturally significant decisions are decisions that affect the architecture of the software system, in some meaningful way. This is irregardless of how software architecture may be defined. These decisions may be in the negative, as in, the decision might

be to *not* do something. The decisions may be explicit or implicit. A decision may have been discussed and a rationale put forth to justify a specific decision and this affected the architecture. Perhaps it was discussed and decided that, while in some ways beneficial, converting the existing architecture to resemble microservices [18] would be a bad thing. On the other hand a decision can be made without anyone realizing. This is what may have happened when the resulting architecture has turned into a big ball of mud (Big Ball of Mud architectural pattern [16]).

For the rest of this thesis architecturally significant decisions are defined as decisions that affect the architecture of the software system in some meaningful way. These decisions shall be considered to be technical for the purposes of this thesis, so business decisions that affect the architecture by imposing requirements and constraints on it are not included in the definition used for this thesis. However, the fact that business decisions affect technical software architecture is worthwhile to keep in mind.

It is worthwhile mentioning the human factor when discussing architecturally significant decisions. What makes architecturally significant decisions significant is people and the context [17]. Many things affect architecture including many decisions, but not all of these decisions are significant or worth spending time on. For instance, a decision to use a particular framework may be very significant in one project but a trivial concern in another because it is such a small part of the project. So in the project where it is a trivial concern it may not be considered an architecturally significant decision whereas in the other project it is. The significance of the decision is also affected by the people involved. Humans are not infallible so even with as good reasoning as possible various psychological effects affect the judgement of significance as well as the decision-making itself. These effects are further elaborated in the next Section.

2.3 Decision-making

Some ways people make decision and what affects those decisions are briefly elaborated on in this Section. First the concepts of rational decision-making and naturalistic decision-making are explained in Section 2.3.1 and Section 2.3.2 respectively. Finally some factors that affect decision-making are described in Section 2.3.3.

2.3.1 Rational decision-making

It is easy to assume decision-making to be rational as it pertains to software architecture design. However, in order to arrive at the optimal solution (ie. being fully rational) in a non-trivial case with 100% confidence an infinite number of alternatives would have to be considered. This is not feasible. This naturally does not mean that it should be assumed that all decision-making in architecture design is irrational, but rather that real decision-making lives on a spectrum between fully intuitive, emotion-based decision-making and fully rational decision-making [12, 30].

2.3.2 Naturalistic decision-making

The way real people make decisions has been extensively studied. There is even a research approach dedicated to this. This is referred to as naturalistic decision-making (NDM) [22]. Decision-making is often far from being fully rational. Emotions are one big factor in decision-making. For instance, decision-making can rely on the emotional reactions to different descriptions of outcomes of the decision. A particular way of describing a potential solution may elicit a more positive emotional response from a decision-maker compared to a description of a different solution regardless of the objective merits of the proposed solutions.

Choosing the first available, viable alternative is very common in decision-making in software design [40, 41]. This means that as soon as the first viable solution is found a decision is reached. Most of the decision-making in our day-to-day lives is like this as well. This form of decision-making puts the least amount of cognitive load on the people making the decision. Therefore it is fast, efficient, and generally good enough [40]. The more structured a problem description is the more likely it is for participants in the decision-making to only consider few options before committing to a decision [40].

A step towards rational decision-making is to consider multiple alternatives before committing to a decision instead of picking the first viable alternative found. This may be referred to as bounded rationality [40]. The decision-makers analyze a number of options and compare them, which may be very rational, but since only a few alternatives are considered, the full array of possible solutions will not be analyzed. This type of decision-making is therefore not fully rational.

This concept of bounded rationality is related to the idea of satisficer behaviour. Satisficer behaviour refers to looking for potential alternative solutions to a problem until a satisfactory solution is found, after which a decision is made [37]. Deciding when a satisfactory solution is found is subjective to the people making the decision. This may depend on a number of things like how critical the problem-at-hand is perceived to be and time pressures.

Most software designers exhibit satisficer behaviour, but there are also exceptions to this rule [38]. Tang suggests that these 'non-satisficing' designers use less analogies, use more reasons for their decisions, and are more convinced of their decisions than satisficing designers. If indeed the majority of software designers and architects behave in this satisficing way, it is worthwhile looking at how these decisions are arrived at and what influences them, since cannot not objectively be known if a decision that has been made is the optimal one.

2.3.3 Factors that affect decision-making

Architectural decision-making, just like any other social activity, is affected by cognitive biases [41]. Examples of cognitive biases that affect decision-making are confirmation bias and framing effect [27, 39]. Confirmation bias states that people are more likely to search for information that confirms their existing beliefs or interpret information that way. Framing effect states that the same information can be interpreted differently depending on how it is presented. Practitioners of software engineering are able to identify biases within decision-making in their own field [41]. However, simply being aware of one's biases does not neutralize them and effort must still be put into minimizing the effect those biases have.

Two cognitive biases conservatism and loss aversion are prevalent in software development related decision-making and can have a large impact on decision-making [42]. Conservatism describes how people insufficiently adjust their views based on new information. Loss aversion refers to people preferring to avoid losses rather than gaining something more. These two biases can occur at the same time which can potentially lead to a large impact in decision-making.

Decision-making may also be affected by seemingly unrelated things like the form of presentation or the order in which different information is received [41]. For instance, information received first may be considered more important than information received

later even though the order may have nothing to do with importance.

A high degree of specialization may lead to a large impact on decision-making [26]. This is because the individuals discussing the decision will have a hard time understanding one another. A potential solution to too much specialization might be to make the resources needed in the decision-making better available to everyone involved in the decision.

Uncertainty in requirements as well as possible solutions also have to be taken into account [36]. How much confidence is there for the requirements being what is actually desired? How likely is it that a possible solution is not going to deliver the desired quality attributes to the extent that is desired?

Decisions do not live in a vacuum. They are often dependant on other decisions and existing context. A previously made decision to use a specific architecture pattern in messaging, for instance, may constrain the options for some other decisions later.

A number of aids exist for helping with decision-making [25]. However, they are not the norm when it comes to making decisions. This may be due several factors; learning a specific method for decision-making requires additional time and effort, alternatives are often difficult to find or express sufficiently etc.

2.4 Agile software development and decision-making within it

In this Section agile software development is briefly explained, including a brief explanation of Scrum because it is the development process, that the process which the case study project uses, most closely resembles. The relationship between agile software development and architecture is also explored, as well as decision-making within agile software development.

2.4.1 Agile software development

Software development refers to the production of software systems. These systems were traditionally developed in a linear, planned-out way. This type of development model is referred to as the waterfall model of software development [32]. Software development according to the waterfall model has some major problems. To rectify these

problems and improve the process of developing software, the concept of agile software development was born. The principles of agile software development include a focus on working software rather than documentation, responding to changing requirements throughout development, and working with the customer continuously [7]. Agile software development generally involves generating working code very early on and then iterating on this code to gradually build larger and more complex systems.

Agile software development is a philosophy or a set of principles rather than a process. From these principles a number of different software development processes have arisen, like XP (eXtreme Programming) and Scrum. A Scrum-like process is employed in the case project of this thesis.

Scrum

Scrum focuses on the idea that there are several constantly changing variables during software development, like requirements, resources and technologies [34]. Software development in Scrum is done in short, usually 1-2 week iterations, called *Sprints*. After each Sprint working software is produced.

In order to cope with the ever-evolving nature of a software project, Scrum includes a number of ”rituals” that happen in every Sprint. The relationships between these rituals are illustrated in Figure 2.1. Sprint rituals are designed to make it easy and quick for the development team to react to changes imposed on them from the outside. The rituals are Sprint planning, Daily Scrum, Sprint review and a Sprint Retrospective. Sprint planning is held right at the start of a Sprint or right before the start depending on how it is viewed. The upcoming Sprint is planned there by selecting what tasks should be done during the Sprint. Daily Scrum refers to a short daily meeting in the morning where every team member describes what they did the previous working day and what they plan on doing today. This is done to help in spotting any potential problems as quickly as possible so appropriate measures can be taken quickly and efficiently. Sprint review happens at the end of the Sprint and it is where the products of the Sprint are reviewed. This usually revolves around demonstrations. Finally a Sprint retrospective happens after the review and it is where the process itself is talked about. This is in an effort to improve the process and make it more efficient. After a retrospective a new Sprint starts with Sprint planning again.

All the work that is to be done is in the Product Backlog. This Product Backlog

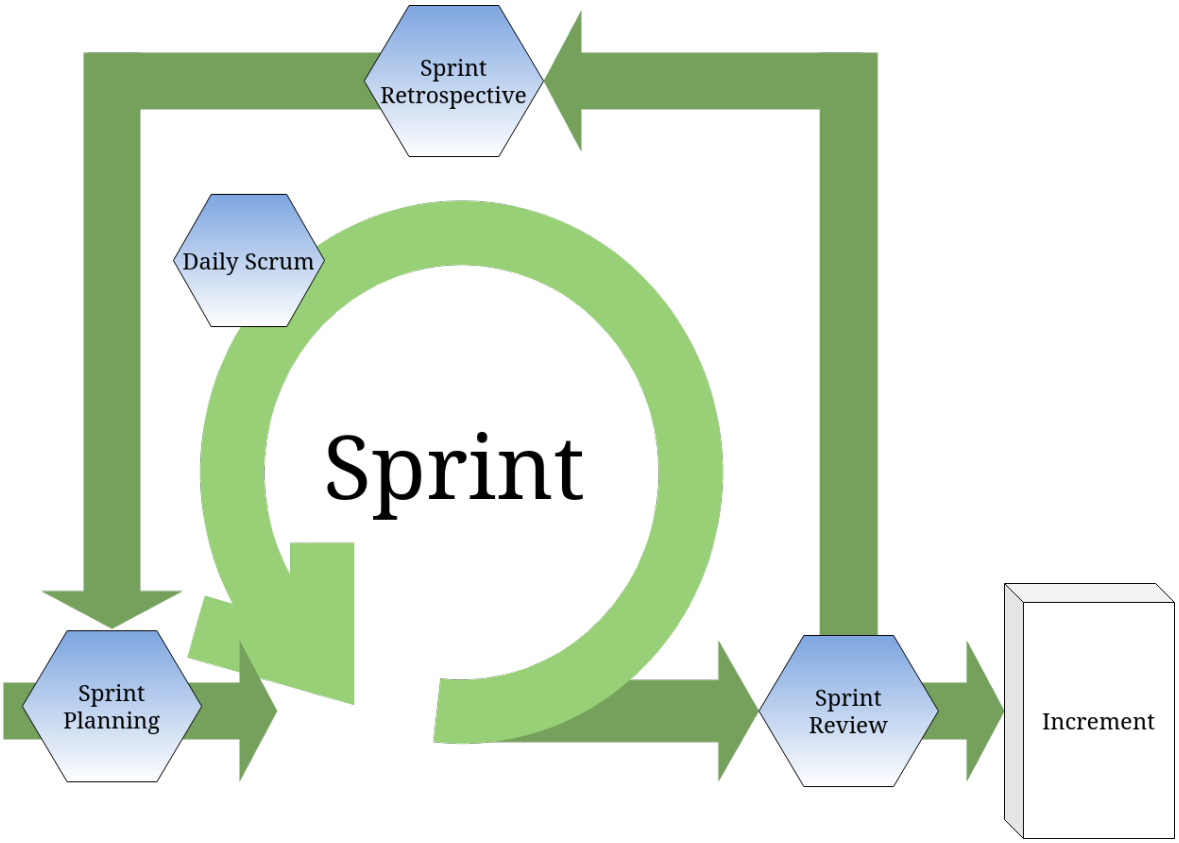

Figure 2.1: Scrum rituals

has to be groomed and maintained so it can be used efficiently. At the beginning of each Sprint, items from the Product Backlog are moved to a Sprint Backlog. The Sprint Backlog contains only the work that is intended to be completed fully during one Sprint. It is the results of Sprint planning.

At the end of each Sprint the result of the completed work is referred to as an Increment. So each Sprint builds on the previous Sprint's Increment and produces a new Increment. The Increment is signed off on at the Sprint Review. The Product Backlog, Sprint Backlog and the Increment are sometimes referred to as Scrum Artifacts. The Artifacts and their relationships to one another are illustrated in Figure 2.2.

There are three key roles that have to be filled in Scrum: development team, Product Owner and Scrum Master. The development team is primarily responsible for building the product, but naturally in order to do this they have to interact with the other roles and various stakeholders. The Product Owner owns the product. They are responsible for communicating to the development team what the product should be and prioritizing development of the product. The Product Owner is responsible for the

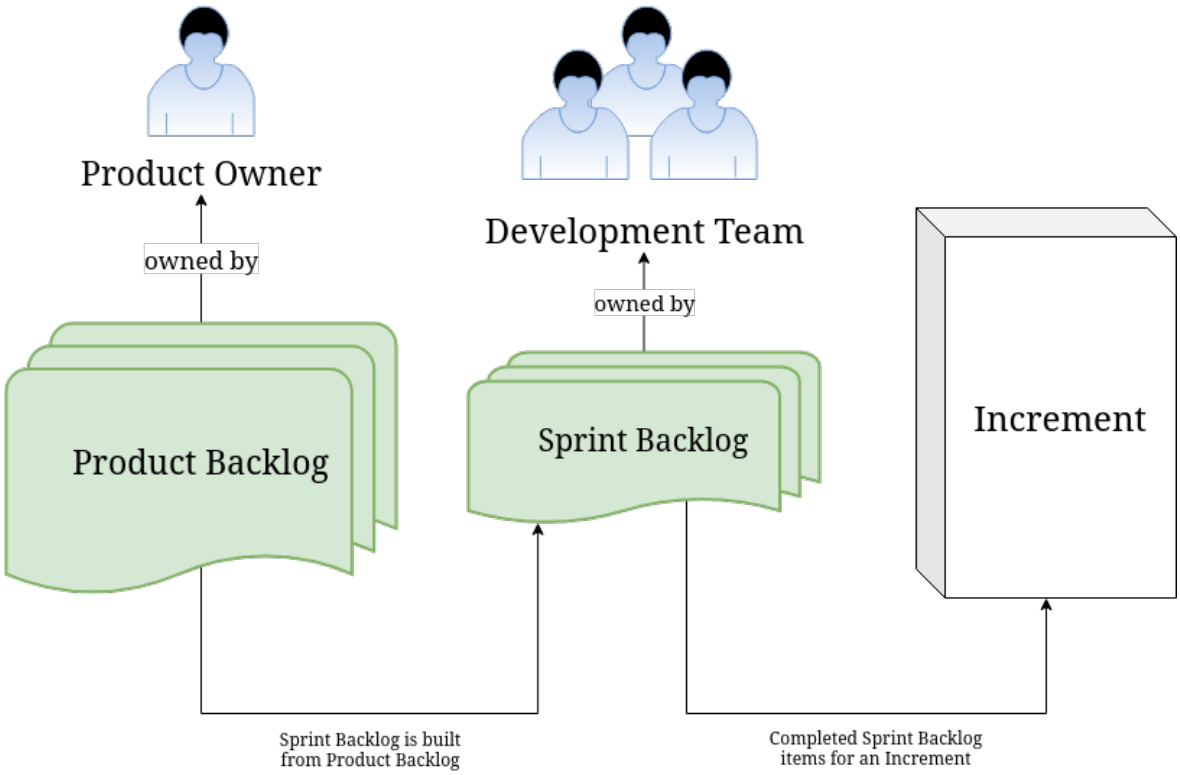

Figure 2.2: Scrum Artifacts

Product Backlog. A Scrum Master is an expert in the Scrum process and is responsible for making the process of developing software as efficient as possible by helping everyone understand the process and trying to make the process more efficient. These three roles form a Scrum team.

There are a lot of details omitted from this description of Scrum, but for the purposes of this thesis this description is sufficient. There are also a number of other agile software development processes, like lean software development and kanban in software development [2, 3, 29]. I will not go over these different agile software development methods because the case project studied regarding this thesis used a Scrum-like development process.

2.4.2 Agile software development and architecture

As agile development is generally against the concept of Big Design Up Front (BDUF) it is not trivial to see how architecture and architectural design fit in with agile software development. When using agile practises architectural planning might often be seen as providing little immediate value for the client [1]. This may lead to little architectural

planning up-front at the start of a project. If not enough architectural planning is done the architecture may become *accidental* [8]. This may, somewhat ironically, lead to similar problems as with too much architectural planning. The poorly thought-out architectural decisions made during development may lead to work having to be done fixing the problems caused by these decisions which in turn leads to reduced production of value to the client. Too much planning may lead to the same problem when requirements change in a way that necessitates a large architectural change and therefore a lot of work done at the start of the project planning the architecture is wasted.

There needs to be some architectural planning up-front especially for complex systems and distributed development in order to maintain a blue print for software development [31].

The role of an architect within agile software development may be that of focusing on the larger scope and context where in an individual project may exist, or that of a development team member who looks at the architecture in the scope of the individual project [1, 19]. In the agile philosophy an architect generally acts in the latter role. The role of an architect that works within a development team can also be taken on by the whole team. Architects who focus on the larger scope may be considered service providers as they often work with several different projects at the same time [14]. These types of architects may even form a kind of a team of their own to make up an architectural board. Naturally there may be both types of architects that affect a project.

These architects may face many problems when working within agile software development regardless of which kind of architect role they fill. Potential problems they may face include busy Product Owners (Scrum terminology), and conflicts both between themselves and the development teams as well as the architects and the other stakeholders [4]. Architects that work outside the development team may also face the challenge of providing sufficient information to various stakeholders efficiently. In Scrum specifically, how architects interact with various relevant entities to further their architectural work is potentially greatly affected by the Product Owner [19]. This is due to how much impact Product Owners have on what is done by the development team and what gets prioritized and communicated to various stakeholders outside the project team.

2.4.3 Decision-making in agile software development

Unlike in traditional, non-agile software development, decision-making within agile software development is made by the development team themselves instead of project managers who take on the role of a facilitator. This type of decision-making within software development teams faces some difficulties as with any decision-making that involves groups of people instead of individuals. Some of these difficulties may include unwillingness to commit to decisions, conflicting priorities, unstable resource availability, the lack of ownership of decisions and the empowerment to make decisions [13]. In order for agile development team members to make decisions they need to feel empowered to make decisions, own those decisions and commit to those decisions when made. In order to combat some of these problems the project manager could take on an altered role somewhere in between the traditional decision-maker and a pure facilitator [24].

Decision-making when it involves agile software development generally happens on several different levels. One way of characterizing these levels is splitting them, in descending order of specificity, into strategic (having to do with organizational planning), tactical (project management related) and operational (implementation specific) levels [26]. These particular levels apply specifically in agile development. In order to make the most optimal decisions possible all of these levels have to be aligned. This means that decisions made on one level must support or enable the decisions made on another level. For instance, if the decision-making on the strategic level does not take into account the realities of the operational level, the decision-makers responsible for the implementation details may have to make decisions that result in too general solutions that do not work well with anything because the strategic decisions lack focus.

3 Research process

In this Chapter the methods employed for the research are explained. This includes the object of the study, the research questions, and the process employed to answer these questions. This Chapter also aims to sufficiently describe the case, including the description of the case company, the case project and the decision that is being studied. Relevant practises employed in the case project specific to this particular case project are also explained in this chapter.

The rest of this Chapter is structured as follows. First the research methods and research questions are presented in Section 3.1. Then the case itself is presented in Section 3.2. Finally the research process is explained in Section 3.3.

3.1 Research methods and research questions

The study conducted for this thesis is a case study [33]. The object of the study is to explore decision-making in the context of an architecturally significant decision made during a subcontracted software development project. In that, this study is exploratory and descriptive. There were no hypotheses before the research was conducted. The theoretical context of the study is provided in Chapter 2. The object of the study is aimed to be reached with the following research questions:

RQ1: How are explicit architecturally significant decisions made during a subcontracted software development project?
RQ2: How are architecturally significant decisions documented?
RQ3: How is it determined that an architecturally significant decision is reached?
RQ4: What factors affect architecturally significant decisions?

RQ1 is specifically about explicit decisions as opposed to implicit decisions that may be made without the knowledge of the people making them.

The unit of analysis is one specific decision. This was decided on in order to focus on getting as much data as possible on the actual decision-making process.

The research process is further explained in Section 3.3.

3.2 Case

This Section aims to sufficiently explain the context of the study by first describing the company and the project during which the decision under study was made in Section 3.2.1. After this, relevant case practises, which refer to specific practises that this specific project had, are described in Section 3.2.2. These practises include what meetings were held that may not be explicitly standard in Scrum-based development. Finally the decision itself is described in Section 3.2.3.

3.2.1 Case company and the project

The case study involves a decision made in a project at Business Finland (referred to as the client) that provides financing for businesses and various other entities [10]. These entities like businesses or universities can apply for financing for their own ventures or joint ventures involving multiple entities working together.

Business Finland has a digital financing service where various entities, like businesses, that need financing can apply for financing. These entities can fill out an application for financing various endeavours. Currently, this application is processed and evaluated by a clerk at Business Finland and a financing decision whether to finance the endeavour or not is made. This process also involves the preparation of the application for decision, after it has been submitted, in order for the clerk to be able to evaluate the application.

It is possible to apply for financing from Business Finland for a joint venture. Joint ventures are ventures that multiple businesses or other entities participate in. Up until now the way financing would be applied for in these types of ventures was for every participating entity to make their own financing application and detail the joint venture in their application. This lead to a lot of work for the clerks handling the applications as every individual application may have described the joint part of the joint venture slightly differently.

Now this old process of handling joint ventures is being renewed and improved. One of the ways it is being improved is that now the service accepts a special kind of application to be submitted by the user called a joint venture application. This application describes the joint venture. The participating parties only need to reference this joint venture application in their own application and are not required to describe the joint part of the venture themselves anymore.

The decision this thesis examines is a part of a project that is implementing the functionality of being able to make these special joint venture application. The project will be referred to here as the Joint Venture Model-project or the JVM-project. As a result of the project there will be support for businesses looking to make a joint venture together in the existing financing systems. The systems where this new functionality is implemented already exist and the JVM-project is part of a larger initiative to renew all of the financing systems.

When submitting a standard application, the application goes from the application portal (web-app for submitting financing applications) to the Registry and from there to the document-management systems. The application goes from the application portal to the Registry automatically, but at the Registry a clerk at Business Finland has to manually review and prepare the application to be sent further into the financing systems for evaluation. Those further subsystems that make up the rest of the financing systems are not relevant to this thesis as the particular decision this thesis focuses on, detailed in Section 3.2.3, does not relate to those further systems directly.

The implementation of the project was subcontracted to a Finnish IT-consultancy firm Solita (from here on out referred to as the supplier). The client and the the supplier have been working together on the financing systems for over ten years now and work together on several projects involving the financing systems. The JVM-project is only one of many projects they work together on currently and includes only a small team of developers. There are several of these small teams from the supplier working on different projects for Business Finland

3.2.2 Case practises

The project where the studied decision was made uses agile software development. They employ a Scrum-like process. Most of the time the development team works in a different city as the client. The development team comes to the client premises once or twice a week for a day. These meetings are referred to as weekly meetings. They start with a status-report at the beginning of the day regarding what has happened in the past week and any new relevant information is shared.

The JVM-project is only one of several projects that involve the financing systems at the client. Since these projects are connected in various ways there are also bi-weekly meetings that involve everyone from the supplier who works on any of the projects

involving the financing systems as well as anyone that is relevant to the meeting from the client. These meetings are done as a remote conferences. They include briefly explaining and sometimes showing what every particular project has achieved recently.

In this particular project it has been common practise to discuss architecturally significant decisions in architecture review board meetings. These review boards include varying people from both the development team and the client. It depends on the agenda of the meeting who attends, but usually at least someone from the development team who is familiar with the architecture of the system as it pertains to the agenda of the meeting, and someone from the client who is familiar with the domain and the context of the agenda attend the meeting. These review board meetings are held on an as-needed basis.

Architecturally significant decisions in this particular projects are documented in an architecture decision record (ADR) [5]. An ADR is a document where architecturally significant decisions are documented. In this project it is a table where each row describes an issue that needs a decision, the decision itself and the rationale for the decision.

3.2.3 The decision

Before the project had started most of the requirements for the project were established by various client stakeholders. One of these was to save the user inputted data from a joint venture application to a document management system as a PDF. The decision that is studied in this thesis is about how to get the information from the joint venture application a user fills in the Application portal to the document-management systems in the form of a PDF. So in other words, where to generate the PDF from the application data and from where to send it to the document-management systems.

Before the decision was made and implemented the joint venture application did not go into the document-management systems and it is required by the client that all applications and documents in general go into the document-management systems. Normal applications had gone to the document-management systems via the Registry, but it was decided to generate and send a PDF of the joint venture application directly from the Application portal. Reasons for this and how this decision was arrived at is discussed in Chapter 4.

3.3 Research procedures

The decision involves several people from both the supplier and the client. Each of the people involved were interviewed. The people involved in the making of the decision in some way were the project team from the supplier and two architects from the client as well as a Product Owner from the client. The list of interviewees is presented in Table 3.1.

The project ran approximately from December to March. The interviews were done primarily in February with two of the six interviews done in the beginning of March. The decision was made by the time of the first interview. All the interviews were one-on-one. All but one of the interviews was done in person. The one interview that was not in person was done over a video call. Two of the six interviews were about two hours long with the other four interviews lasting approximately one hour each. Two of the one hour interviews had a relatively strict one hour time-constraint due to the busy schedule of the participants.

The interviews were semi-structured. There was an agenda for the interview and a set of guiding questions. Almost all of the questions were open, but there were a few closed ones as well. Therefore most of the data collected was qualitative in nature, but there were were also some data that was quantitative.

The time-constraints on completing the research made it so that there was no possibility to do iterations of interviews. There was, however, some flexibility within the set of interviews. Some questions were tweaked in order to better get the idea of what was being asked across based on the previous interviews upon learning more about the case. Some questions were also added based on new unexpected information that arose in previous interviews.

Title	Role	Description
Software architect / developer	Supplier's architect	A developer and an architect with over ten years of experience with Business Finland and more as a professional
Software developer	Development team member (supplier)	Couple years of experience with Business Finland. Roughly a year full-time
Software developer	Development team member (supplier)	Several years of experience as a professional developer. Couple of years with Business Finland
Product Owner	Product Owner from Business Finland	Approximately 20 years of experience in the domain. Has been with the JVM-project from the start. Deals with domain-related concerns and requirements and acts as a conduit between the development team and various stakeholders at Business Finland.
Enterprise architect	Architect from Business Finland	More than ten years of experience with the whole enterprise architecture and the technical architecture of the whole. Not a member of the project team.
System Architect	Architect from Business Finland	Several years of professional experience relating to software architecture. Only a few months of experience with Business Finland at the time of the interviews. Not a member of the project team but works with the team on architecturally significant issues on a regular basis when needed.

Table 3.1: Interviewees

4 Results

The project in question deals in creating a way for businesses to finance their joint ventures with other businesses. This requires a new type of application (joint venture application) to be submitted to the system that is different from a normal application for financing. For one, there is no financing decision made on it. It only describes the joint venture. Because there is no decision made on this joint venture application, it has to be handled differently within the system. One requirement when implementing the new type of application was to add a PDF of the joint venture application to the document-management systems so it can be referenced by other applications. Considering the existing architecture of the system a decision had to be made regarding where and how this PDF is generated and exported to the document-management systems. This project-level architectural decision is the focus of this thesis. More details on the case can be found in Chapter 3.

In this chapter the results of the case study are presented. First the options that were considered as potential solutions are described in Section 4.1. Then the decision-making process is described in Section 4.2. After this the rationale for why the given decision was made is explained in Section 4.3. Finally, Section 4.4 describes further observations on why the decision was made the way it was, and how the process went. The primary data gathering method was interviews.

4.1 Options for the solution

The problem of getting the information about a joint venture application to the document-management systems as a PDF was researched in the course of making the decision regarding the problem. This research produced two clear alternatives, referred to here as Option A (depicted in Figure 4.1) and Option B (depicted in Figure 4.2). These two options were not the only two possible solutions, but they were the only ones that the development team considered potentially viable at one point or another.

With Option A the PDF is generated and exported directly from the application portal as portrayed in Figure 4.1. The blue line in the figure represents the path the PDF, that was made from the joint venture application, took to reach the document-management

systems. This option requires a new integration between the Application portal and the document-management systems.

With Option B the joint venture application is transferred to the Registry like other, normal applications and then there a PDF is generated and moved to the document-management systems. From the Registry the PDF would have to be manually moved to the document-management systems. This option is portrayed in Figure 4.2. Again, the blue line represents the path of the data from the joint venture application that needs to go into the document-management systems in the form of a PDF. Joint venture applications are not the same kind of applications as other applications so the Registry would have to be adapted to handle these different types of applications.

The figures describing the potential solutions are made based on the interviews and Antti Arela's existing thesis work relating to the systems [6]. Option A was the selected solution that has been implemented and went into production.

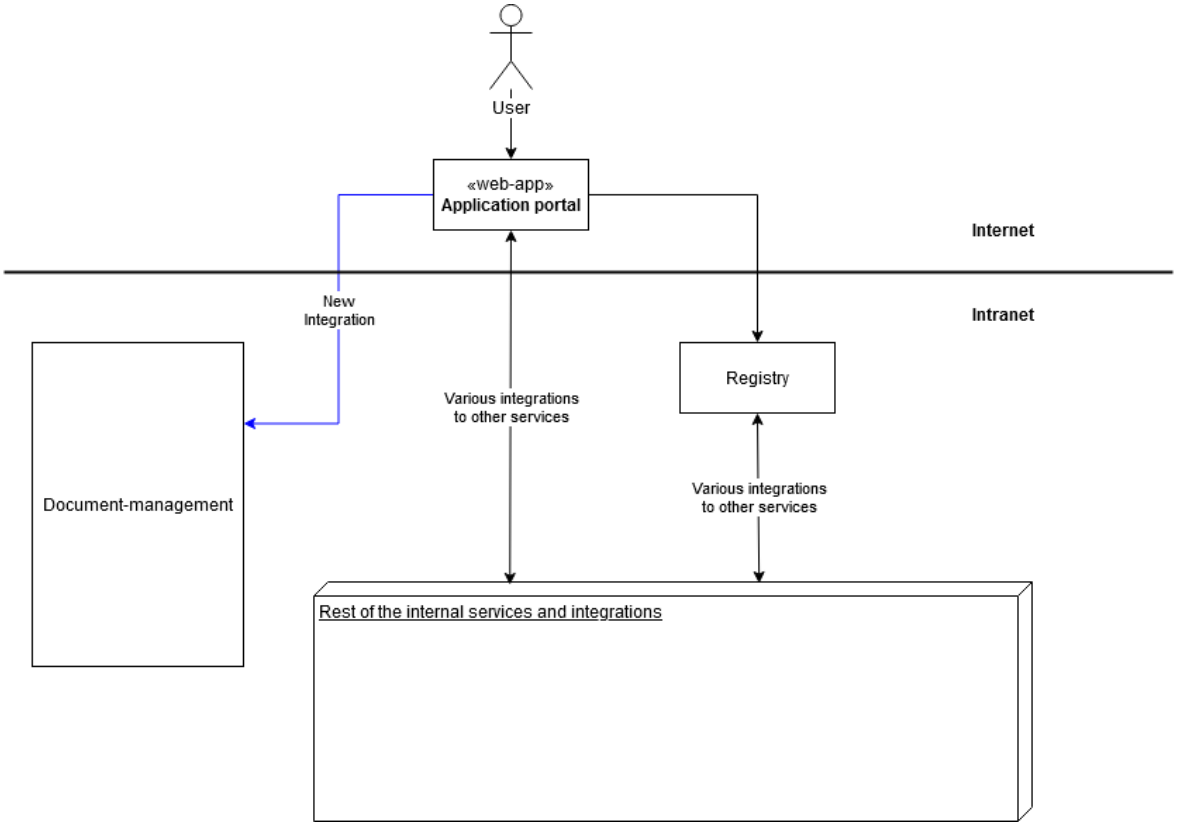

Figure 4.1: Option A. Direct integration from application portal to document-management.

These two options were the only ones that were given proper thought and analysis. It is noteworthy that even though both options were considered at one time or another, there were people who were confident that moving the joint venture application to the

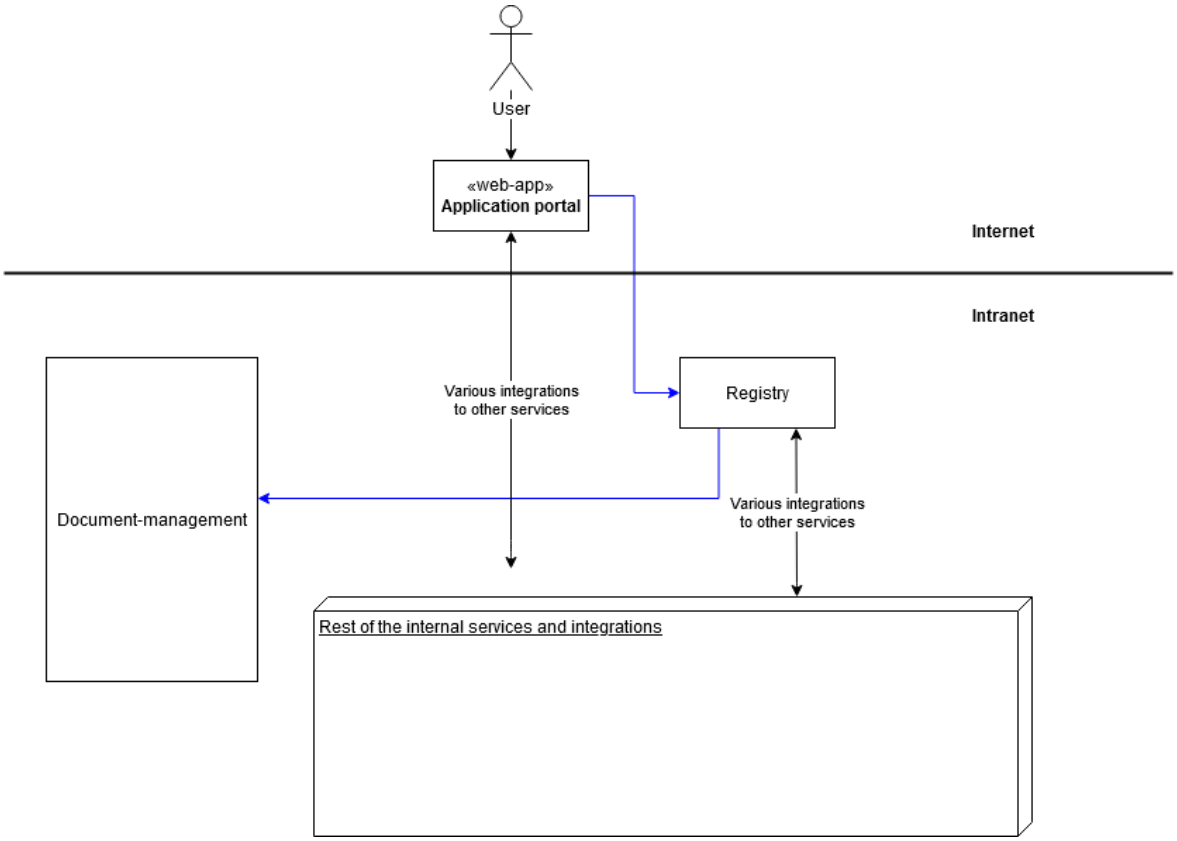

Figure 4.2: Option B. Joint venture application is sent to the Registry where a PDF is generated and manually exported to document-management.

Registry was never a good option because the Registry is old and there are already plans in motion to remove that app completely from use. Others were not as sure how the problem should exactly be solved so. For instance, an architect from the client side, who had extensive knowledge on the system and therefore had a good idea of what should not be done, was certain the Registry should not be used because it will be obsolete very soon. Of course it is not wise to blindly believe one person, but had the development team known the Registry is very likely not a good option, perhaps the work spent on researching the problem could have been lesser. Knowing the opinion of said architect would not have negated the need for researching the problem but it might have guided and informed the research thus reducing some work.

Naturally it is not always easy to know who knows what and when one should express their thoughts on things as ”information overload” may also become a problem. If there is too much information it is hard to prioritize and remember it all. The lack of communication on this particular topic was mainly due to lack of time on the part of the architect on the client-side. This is a known problem at the client and had been

addressed at least partly by bringing in a new architect to take some of the load off the architect.

Beyond these two options, one other potential option was briefly mentioned in one interview. The joint venture application could've continued further down the pipeline, the same way a normal application would, into the financing-ERP. This option, however, was dismissed off-hand and not considered further.

4.2 Mining the decision-making process

The requirement within the project was to have a PDF of a joint venture application stored in the document-management. This requirement was a part of the initial requirements for the project so it was known from the very start of the project that a PDF of a joint venture application has to be generated and exported to the document-management systems. Some of the people that were aware of the requirement knew early on that an architecturally significant decision regarding how this requirement is implemented will need to be made at some point. A pressing need for a decision regarding this requirement arose around late December when the implementation task of generating and exporting a PDF of the joint venture application was initially looked at by a developer. The problem and potential solutions were researched after the need arose and a proposal was formed and shared. The proposal was in the form of an ADR-entry. By the next day this proposal was deemed good, and development relating to the decision was started the next week. At this point the client had not fully signed off on the decision. The decision and its reasoning were formally introduced and explained to the client's architects and PO on 30.1. although brief mentions of the decision may have been mentioned earlier in passing. There the decision was signed off on by the client and the decision was not discussed or thought about anymore after that at least as it directly related to the project. The timeline for the decision-making is outlined in Figure 4.3. It also includes the people who affected the decisions.

When it came to doing the research required to make a justified decision, one software developer from the development team compared different optional solutions to the problem and wrote down reasons why the PDF of the joint venture application should be generated and exported directly from the Application portal and not be generated and exported from the Registry. This reasoning sketch was deemed good by the project architect who then added it as an entry to the ADR. The entry included the proposed

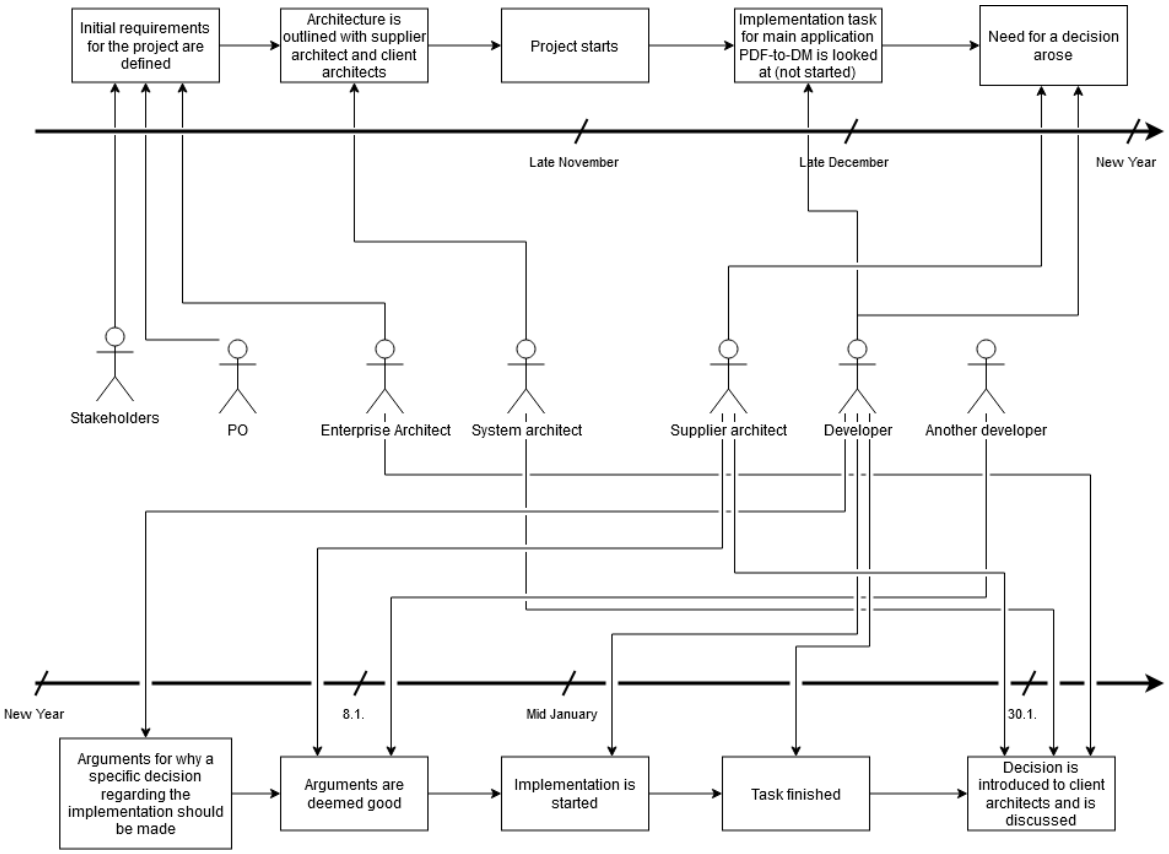

Figure 4.3: Timeline for the decision-making process

decision and the reasoning for it.

The issue relating to the decision was brought up in some weekly meetings (as mentioned in Section 3.2.2), both when it was worked on, which sparked some conversation about it, and when the implementation was done. The decision was also brought up in a bi-weekly meeting involving all the different projects that work on the financing systems. In that meeting it was only presented and no discussion was had about it.

The decision in question was also brought up in an architecture review board meeting late January. At this time the decision was already made in practise and implemented. This was the meeting where the decision was finally signed off on by the client. So as the name suggests, in the architecture review board meetings the architecture is reviewed, signed off on, and discussions are had relating to the architecture. Often when decisions that are already made and are being introduced in the meeting they spark conversation about something else that is still in some way related to the original decision. These discussions may lead to finding out new architecturally significant decisions that need to be made. In these architecture review board meetings decisions are not generally

made, but rather reviewed and approved.

The decision-making process in this project has not been documented beyond some meeting notes and the focus with those notes is not necessarily to aid in decision-making. As mentioned before, the decision itself was documented in the form of an ADR-entry. There were some varying opinions on whether the decision-making process should be documented as it is happening or whether or not there should be some sort of a process or a ”playbook” in place for how decision-making should be done. Some interviewees did state that a type of ”playbook” where more general practises and guidelines for decision-making are documented might be useful. These would include things like who to ask, when to ask and where to ask in specific situations like if a particular decision would require a new integration to another system. It might, for instance, help with figuring out who to ask when a question arises.

The most time spent relating to the decision in question was spent by the developer who researched the different options for the decision. They spent approximately one or two days on researching the problem. The rest of the relevant people did not spend as much time on it. It was estimated by one of the interviewees that they (the individual interviewee) spent approximately half an hour in total on the decision by discussing it in two separate and reviewing the written rationale for the decision. The rest of the interviewees expressed similar sentiments.

This decision was made alongside software development. Most of the other architecturally significant decisions made for this project were made before the development began. So this particular decision was special in that way that it had to be done alongside normal agile development which was done using a Scrum-like process. The project had no explicitly documented process for handling decisions that involved people outside the project team beyond bringing relevant issues up in weekly meetings. The decision was brought up and discussed in a weekly meeting along with the rest of the relevant issues that had to be discussed in those meetings relating to furthering of the development. The research process of the decision-making was done as a part of normal Sprint development by a developer, but the final approval given by the client was outside the agile development process as it was done in an architecture review board meeting. So in practise most of the decision-making was integrated into the agile development process, but the final review by the client was not, although the organizing of required extra meetings, like the architectural review board, was done within the development process mostly within weekly meetings. So another way of

characterizing the decision-making that involves people outside the project team is to think of it as an appendix to the development process that is appended to the side of the process when required.

Since this specific decision was clear and could be researched by a single developer, it did not require a lot of iterative communication between the supplier and the client but generally decisions that are architecturally significant do go through iterations of conversations between the supplier and the client. There was a sentiment from one of the interviewees that sometimes the conversations may also be had outside the scope of the supplier and therefore create uncertainty as the supplier has no visibility to the client outside the systems they are developing and what is required for that development.

The product owner acts as a conduit between the development team from the supplier and various relevant stakeholders on the client-side. They make sure the right people from the client are present when needed on behalf of the development team. They are also the naturally the primary source of guidance for the development team for things relating to the domain and business-side of the project and its context.

Here is a brief, generic summary of the decision-making process for this particular decision containing only the main points. (1) A need for a decision is discovered. This originated from a requirement imposed on the development team. (2) The problem that required a decision was researched by the development team and a proposal was put forth. (3) The proposal was tentatively accepted by immediately available people (ie. some of the other development team members). (4) Decision was implemented. (5) Final approval was given by the architectural review board.

It is important to mention here that even though the decision was made and acted on before the final okay from the client, the development team and the supplier in general enjoy a lot of trust from the client. They are able to and even encouraged to make technical decisions quickly when it is required. This is likely due to the nature of fast, iterative agile development processes in comparison to generally more rigid and slower business processes that require several different stakeholders to work together. The trust between the client and the supplier, at least in part, is likely the result of a history of working together spanning over a decade.

4.3 Rationale

There were several considerations that affected the decision turning out the way it did. The main one was workload. Choosing to take the joint venture application to the Registry and then from there to document-management systems would've required roughly double the work compared to generating and exporting the PDF directly from the Application portal. The emphasis on the amount of required work was motivated by a tight schedule for the JVM-project. The tight schedule was not an issue for any of the interviewees although a few of them did admit that a more lax schedule would have lead to more analysis and considerations surrounding the architecture and decisions regarding it.

There is also a plan in place to completely replace the Registry in the future. Therefore it makes little sense to spend a lot of time developing on this old system that will soon be taken down. This, however, was not a primary concern for the development team when considering the different options. This is not to say it was not considered by the development team, but rather that it was not a primary concern. This concern was more clear and important for the client.

Another important consideration was security. In this particular case it did not dictate the chosen solution because there were no security concerns found about making the decision either way between the two main options. Some effort was spent on discussing and thinking about the security aspect though. The fact that the chosen solution (Figure 4.1) required a new integration was brought up in several interviews as a potential security concern. This potential concern was dismissed because there already exists several similar integrations to the document-management systems in the whole financing-system.

4.4 Further observations

There was a clear consensus among the interviewees that this particular decision was made generally well. No major complaints from any of the interviewees. One interviewee did point out that the final approval on the decision could have been given earlier, as there was more than three weeks between the rationale of the decision being written and made available for review and the decision being discussed and finalized with the client (see Figure 4.3). The same interviewee did understand the reasons for

this as the relevant architects on the client are extremely busy. The lag in the final decision being made did not adversely affect the development, although conceivably it could have. Had the decision been that the chosen option to make a new integration (see Figure 4.1) was not acceptable, it would have caused a lot of extra work for the development team which might have made the schedule difficult to keep.

There were some differences between the supplier and the client on how they viewed the people who most affected the decision. The client does not have visibility to the inner workings of the development team and how they operate since the development is subcontracted. For instance, on the side of the client there is a prevalent impression that the architect from the supplier was the primary influence on how the decision came out, but most of the people from the supplier, including the architect themselves, feels it was one of the developers, who did the research on the different potential solutions, who had the biggest influence.

Several interviewees from both the supplier and the client also expressed a believe that the architect from the supplier had a good idea of what the decision should be, but the architect themselves was not sure how it should be until arguments for a particular solution were presented by a developer as mentioned above.

There was a clear difference in how developers and other stakeholders, including the Product Owner and client architects, experience the timeline of the decision-making for this particular decision. Client-side stakeholders estimated the timeline for various events in the timeline to be later than the development team. This is partly due to different perspectives and the scope at which different stakeholders see the decisions. For the supplier the decision is within the scope of the project; there is a requirement which has to be filled in some way during the project. For the client the scope is bigger in that they also consider why the requirement exists and how it came to exist as well as the larger architecture of the whole collection of systems at the client. This is not to say the supplier is necessarily unaware of the reasoning for the requirement or the larger scale architecture, but those concerns are outside the scope of the individual project the supplier is focusing on.

The development team made a decision early on in the beginning of January and they were confident in it and that it would be final, but on the client-side the Product Owner, for example, did not consider the decision to have been made until early February. This again relates to the scope at which the client operates in compared to the supplier as well as differing perspectives. Especially non-technical stakeholders, like a Product

Owner, cannot determine if a decision is final until they truly see it working in action. In a large organization a decision is technically not final until it is signed off on by the very highest authorities responsible for the systems possible. This was also mentioned by the Product Owner. However, there is also trust between different levels of management within the client organization just like there is trust between the client and the supplier to make their own decisions and make them well.

It might be expected that nothing architecturally significant, like building a new integration between two systems, should be implemented before the client has decided it is the right thing to do, but the fact that this project is done as a tightly scheduled, subcontracted project means that the development team cannot wait around for a final decision to be made before beginning work implementing it. Not breaking the schedule may be more important to the supplier as they have agreed to delivering software into production by a certain date. This is a necessary trade-off between staying within schedule and keeping the client in charge of the important, architecturally significant issues. The client does have trust in the supplier to make these preliminary decisions which indicates awareness from the client that the development cannot wait for final approval to actually start work on something.

There was some preliminary messaging done regarding the decision and its requirements in a text chat between the supplier's team members and the client. These messages were sometimes overlooked or missed. There was a sentiment from some of the interviewees from the supplier that this is a somewhat common occurrence and it is often required to bring things up in person if those things are important. This may have potentially somewhat slowed down the decision-making, especially the research part of it. It is worthwhile to note however, that questions asked in a text chat might not be asked in person at all and being close enough to constantly be able to ask someone about a certain thing is unfeasible as it would likely require constant presence within the same work area by both the client and the supplier. Especially for more business oriented people this is likely not an option at all especially considering how much they might have to move between places for meetings and such. There are also potential benefits for using a text-chat alongside face-to-face meetings as text-chat enables a large variety of people to participate in the conversation and leaves a log of the conversation. This was brought up by one of the interviewees.

None of the interviewed parties on the supplier side were sure of what the decision should be prior to researching the different options. However, both the enterprise

architect and product owner from the client-side expressed that it seemed obvious to them that Option A (see Figure 4.1) is the only sensible solution. They expressed that this was because they were very well aware that the Registry will be completely torn down in the near future.

A problem this project did have when it came to issues that required input from the client architects was that these architects were incredibly busy. Especially the enterprise architect had a lot of work on their hands. Some interviewees even consider the enterprise architect to be overworked. This problem is being worked on. It did, however, impact many of the points mentioned above, like the final approval taking a long time to give. It could have also saved some development time and effort had everyone in the development team been aware enough about the confidence the client had on Option A (Figure 4.1) being the correct one. The developer who researched the different options may not have needed to spend as much time researching the different options had he been aware enough.

As mentioned in Section 2.3 the more structured a problem is the fewer options people tend to consider. This specific, studied decision is indeed inline with that statement. The problem was very structured: information about a joint venture needs to be somehow extracted from an application and put into the document-management systems as a PDF. Only two options were considered even though there were more at least conceivable options to consider. This particular decision-making process also fell within what is described as bounded-rationality, meaning more than one option was carefully considered using rational reasoning, but the full spectrum of possibilities was not considered. This made the decision-making bounded-rational and not fully rational. As mentioned before, at least one additional option beyond the the two mentioned (Figure 4.1 and Figure 4.2) was thought of but dismissed off-hand.

There was an implementation task for the development team in the project management tool, Jira in this case. The task was named in such a way that at least to one of the interviewees it implied that it should be architecturally implemented in a specific way. The implication to this particular interviewee was that it should be implemented by exporting the PDF directly from the Application portal. This happened to be the chosen solution, but the task itself was ambiguous enough that further research had to be done to figure out what the correct solution is. This may be considered a case of a cognitive bias called the Framing Effect [39]. The framing effect refers to the effect the presentation of a problem has on the solution. In other words, presenting then

same problem in a specific way may lead to specific solutions that might be different if the problem was presented in another way. In this particular case, the implication from the title of the task in Jira happened to be the same as the solution that was arrived at after research and analysis of different options, but in a different case with less research the resulting decision might have been sub-optimal.

The interviewees did not, however, recognize any cognitive biases in relation to this decision or the project in general themselves. There was also no indication as an outsider, based on the interviews, that there were cognitive biases that affected this decision-making process in a particularly meaningful or noticeable way. The interviewees did, however, recognize having noticed a small amount of cognitive biases (when the biases were described) in general within their work over the years, but could not describe any instances in detail.

5 Discussion

In this Chapter the results are analyzed, some perceived problems pointed out, and some generalizations made. This Chapter also contains discussions on validity threats that may have affected the research described in this thesis. The Chapter is structured as follows: Section 5.1 contains the analysis of the results and Section 5.2 elaborates on the threats to the validity of this thesis.

5.1 Analysis

There may be excessive load put on the enterprise architect of the client, which drags out finalizing of some decisions. There is another, newer architect at the client that was involved with this project, but they were not taking as active of a part in the discussions at the time because they were still not familiar enough with the architecture and context to have strong opinions. This is to be expected of a new employee coming in to work on something very complex. This newer architect had been involved with the systems the JVM-project relates to for about six months at the time of the interviews. They also seemed to not hold as much esteem when it came to the architecture of the overall systems. This is likely because people are used to going to the enterprise architect who has earned their trust over the years. A new architect coming in to in part replace the old architect has a naturally difficult job of gaining people's trust. In other words, the client-side enterprise architect was overworked because people go to them with problems and questions relating to the architecture. To combat this, the new architect was brought on-board, but the load has not divided evenly among them, at least not yet. This was likely at least in part due to the fact that the new architect is less experienced in the architecture of the overall systems. This highlights the importance of efficient familiarization of new project members.

It is difficult to say based on interviews alone whether the people involved in the decision-making were exhibiting satisficing behaviour or not. On one hand, several options were considered by a developer and a rationale was formed and written down which would indicate non-satisficing behaviour. However, it is difficult to determine based on interviews alone what the thought process of any of the people involved was

like. Even if the one developer that formulated the rationale for the decision was exhibiting non-satisficing behaviour, what about the rest of the involved people. Most of the other people involved spent very little time on the decision suggesting that they behaved in a more satisficing way. This is not necessarily a bad thing, as energy spent on the decision may be wasted and it could be better spent elsewhere. Perhaps it is an efficient, but still a rather safe way to make decisions by offloading the rational part of the decision-making to one or few people while the other relevant people only review the rationale. This way not everyone needs to spend a lot of energy on the problem, potentially doing the same work as someone else is figuring the problem out. Naturally this will not guarantee the most optimal solution, but it does strike a balance between finding the most optimal solution and efficiency.

This way of splitting work for a given decision may perhaps be generalized to decision-making relating to software development in general; one or a few people do the bulk of the work by researching the problem and proposing a solution and other relevant people review the proposal or the findings of the research to make the final decision. The initial proposal goes through people close to the people researching it first as it has the lowest threshold and from there continues up through the relevant people that need to review it. This may be a very efficient way of making decisions as it frees every apart from the people researching the problem free to spend their time and energy on other things.

The fact that the project uses a Scrum-like development process means that the decisions that arise mid-project can more easily be handled, because there already is a process that is designed to handle changing requirements and surprises. The decision was researched and decided on within the project team very quickly as there was enough communication and periodic meetings to facilitate this quick decision-making. However, from the point of view of the client, not counting the Product Owner, the final decision-making did linger on almost a month after the decision was made in practise by the development team. This is likely mostly due to scheduling difficulties and high workloads on various relevant stakeholders. Had the decision been rejected by the client stakeholders in February after it had already been implemented it might have cost a lot of additional time and therefore money for the client to remove or change the existing implementation and re-implement it in a different way. The requirement to have the relevant information from the joint venture application in the document-management systems in the form of a PDF was an important requirement

after all.

It is hard to say whether or not there is something that can be done about this issue of business related stakeholders lagging a bit behind the technically side in terms of decision-making. Perhaps it might be possible to communicate with the business stakeholders in an asynchronous manner as soon as possible in as efficient of a way as possible in order to get the final approval as quickly as possible to reduce the risk of having to do double the work due to having to implement and then re-implement the same thing. But how much of this kind of approval can be done asynchronously? Could all of it be done that way? After all, this particular decision did not produce a lot of discussions in the meetings it was discussed. Perhaps if it is possible to identify these types of decisions where one person or potentially a small few can research the decision efficiently, make a proposal, agree to it locally (so in this case within the development team as the communication within the team was unsurprisingly efficient), and submit it for review to the relevant stakeholders. Of course identifying the types of problems and decisions that can be made this way is likely very non-trivial and requires the stakeholders that review the proposal to remember to review the proposal as soon as they are able.

Efficient communication about technical decisions to non-technical stakeholders is not a trivial task. The fact that this project is a subcontracted project further complicates the communication issue as the developers from the supplier will likely never be as familiar with the enterprise architecture and business processes as people from the client. I would even argue that developers from the supplier should not need to be as familiar with the business processes because learning these processes is going to take a lot of time and working efficiently is an important concern for the supplier. The turnover on a development team from an IT-consultancy firm might also be relatively high which makes it even more important that time is used efficiently.

Since the interviewees agreed that this particular decision was made well and the process of making it worked at least reasonably well, it could perhaps be used as a model for future decision-making of this nature or as a prototype for building a ”playbook” for decision-making. Naturally if some form of a structured process is implemented for architecturally significant decision-making alongside agile software development it should not be too rigid as this would likely not work well with the agile nature of the software development. A set of guidelines and tips on how to go about advancing an architecturally significant issue when it arises might be helpful to reduce cognitive load

on the people involved. Especially developers would likely benefit from this as they would not need to spend energy on trying to figure out the best course of action on advancing the issue.

There is a lot of trust in the very experienced supplier architect. This brings with it a sense of comfort for the client that the decisions he's involved in are likely to be good. Various people on the client-side do question decisions and their reasoning, but a sense of comfort may still lead to sub-optimal decisions if less thought-out decisions are more likely to be adopted due to that sense of comfort. There is nothing to suggest that anything was done in a sub-optimal way because of this comfort regarding the studied decision, but it is worthwhile keeping in mind because there was clearly a great deal of respect for the abilities as well as the opinions of the supplier's architect.

It was mentioned by some of the interviewees that questions raised in the text-chat, that is shared between the development team and the relevant people from the client, tend to sometimes get lost in it and never responded to. This may cause some frustration for the people asking the questions as they have to deal with the uncertainty of not knowing something they need to. Naturally the most important questions that absolutely require answering will be asked and answered in person, but for one, seeing and finding the right person in person might not always be possible or easy and secondly questions asked in a text-chat are not always questions that absolutely require an answer quickly. Sometimes these questions that are asked in a text chat may be such that if they were answered it could make work for the people asking the question more efficient. The efficiency may come from the answer just solving a small problem or clearing something up, but it may also come from taking away cognitive load from the questioner allowing them to not spend any energy on the question whether they realize it or not. The reason for messages sometimes getting lost is in large part affected by how busy some people are. The culture regarding text-based messaging may also affect how likely messages are to get lost as well as overuse or poor use of chat (ie. sending too many messages or sending them in the wrong place or to the wrong people). Based on the interviews it is not possible to say if culture had any effect on messages being lost in text-chat for this particular case.

Chat has a benefit over face-to-face meetings in that it leaves a log of communication and allows everyone to participate and at least see what is being discussed. For these reasons it is considered by the project team to be a valuable part of the overall communication in the project.

People might also ask things in a text chat that they might not otherwise ask, perhaps because they deem the issue to be too trivial to take up actual face-to-face meeting time. Regardless, the questions, even if somewhat trivial, might make the job of the person asking the question easier by saving them time from figuring it out by themselves or by them not needing to spend any time thinking about how to advance the issue, thus reducing cognitive load.

5.2 Threats to validity

Only one decision was studied for this thesis. This reduces the generalizability of the results somewhat. Perhaps this particular decision was done in an unusual way even within the scope of this project.

One interviewee pointed out that this specific decision was particularly firmly decided. There was a unusually large amount of confidence in the decision. If indeed this particular case was an oddity, it further reduces the generalizability of the results.

Everyone that affected the decision directly were interviewed, but observer triangulation may have been improved slightly by including people who indirectly affected the decision, like various business stakeholders, in the pool of interviewees.

Because this study focused on interviews as the source of data, the level of data triangulation was not very high. This could be improved by including observations of a software project like the one studied in this thesis along with interviews and potential documentation.

The interviews were conducted in a span of about a month. This was primarily due to scheduling difficulties and the fact some interviewees had incredibly busy schedules. A month is still a relatively a long time and new information or results may have come in during that time or at the very least people's opinions may have changed or shifted slightly in that time. This has some impact on the validity of the study.

Another threat to validity is that even though the decision was made relatively recently, between Christmas to early February, memories of the process have already somewhat faded. It is also possible some memories have, to some degree, warped or distorted. In this respect it would beneficial to do further research into decision-making processes by means of observation; being present during a project with the team and in meetings to see how the process actually unfolded.

6 Conclusions

This thesis analyzed architecturally significant decision-making in a subcontracted software development project through the means of a case study using interviews as the primary source of data. The studied decision was explicitly known to the people making it and it was carefully considered by a number of people. This is opposed to a decision made without explicit considerations which would be another type of decision-making worth further research elsewhere.

It was generally known at the start of the project that an architecturally significant decision regarding how to generate and transfer the PDF containing the relevant information about a joint venture would have to be made eventually during the project. When an acute need for a decision arose the decision was quickly handled by the development team within their development process and a preliminary decision was made. The final client-side approval of this decision did lag behind somewhat. This was in large part due to scheduling difficulties and how busy the relevant stakeholders were. This delay in the final approval being given by the client highlights the difficulty of incorporating people from outside the development into an agile software development process.

The decision itself was documented in the form of an ADR-entry, but the process of making the decision was not. Most did agreed it might be helpful to document the decision-making process as well if done well, but potentially even harmful if done poorly resulting in increased rigidity and slowness of the development process.

In practise the decision was reached when the development team and especially the supplier's architect agreed that the rationale for the decision was sound. However, from the point of view of the client, the decision was not reached until it was reviewed by the enterprise architect in an architecture review board later when it was already implemented. These two viewpoints differ in their timings. The final approval from the enterprise architect came several weeks after the decision was made in practise. The development team could not have waited that long or waited at all really as they had a tight schedule to keep in the completion of the project. The client did however express trust in the supplier to make these kinds of decisions on their own in advance when needed. This also highlights the differing scopes of reasoning of the supplier and

the client.

Overall the decision-making was considered good by the interviewees and there was not much that could have been improved in their eyes. A slight problem, however, was that the text-based communication between the development team and the client was slow at times. If things were not asked in person, they were sometimes not answered at all. What makes communication worse yet is that the relevant people on the client are very busy. However, as mentioned before, the development team enjoys a good deal of trust from the client in their decision-making when it comes to technical decisions. This works well in combating any slight issues and delays caused by unavailability of the relevant client personnel. The decision was handled quickly and efficiently apart from the text-based communication slowness and getting the approval from the client.

Several things affected the outcome of the decision-making process. The main one was the required workload. This was due to a tight schedule. The decision would likely have been made roughly the same way even if the schedule was a bit more lax, but in the considerations of the decision the fact that the chosen option required the least amount of work was the most important. The schedule was not an issue for any of the people involved.

The client had slightly differing priorities when considering the decision compared to the development team. It was very important for the client that the decision be done the way it was because the other option would lead to even more work in the future due to the system it involved being taken down in the near future. The development team was aware of this, but did not emphasize it as much as a rationale for the decision as the client did.

Security was another consideration but the security concern was acceptable because another similar concern is already in place in other systems with the client and has no issues.

The work that went into the decision-making was mostly done by one developer who researched and prepared the rationale for why the decision should be made a certain way. Other people that affected the decision mostly actively worked on the decision only in a handful of meetings. Naturally several people had an indirect effect through earlier decisions and conversations not directly related to the decision.

A generalization may potentially be drawn from how the work on the decision was split among different people and how the information about the decision spread to

other people. A small number of people research the problem and propose a solution. This in turn is reviewed in iterations by relevant people starting from people close to the researchers and continuing through other relevant people. This is essentially how the decision-making went in this case study starting from a developer researching the decision, writing a proposal that was then deemed good by the supplier's architect who then later described it to the client side architects who approved it. This general process worked well in this case as it was non-blocking in terms of overall software development and only involved a single developer most of the time.

Bibliography

[1] P. Abrahamsson, M. A. Babar, and P. Kruchten. "Agility and architecture: Can they coexist?" In: *IEEE Software* 27.2 (Mar. 2010), pp. 16–22. DOI: 10.1109/MS.2010.36.

[2] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. "Agile Software Development Methods: Review and Analysis". In: *VTT publication* 478 (Sept. 2002), p. 112. URL: http://arxiv.org/abs/1709.08439.

[3] M. O. Ahmad, J. Markkula, and M. Oivo. "Kanban in software development: A systematic literature review". In: *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*. IEEE Computer Society, 2013, pp. 9–16. ISBN: 9780769550916. DOI: 10.1109/SEAA.2013.28.

[4] S. Angelov, M. Meesters, and M. Galster. "Architects in scrum: What challenges do they face?" In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9839 LNCS. Springer Verlag, Nov. 2016, pp. 229–237. ISBN: 9783319489919. DOI: 10.1007/978-3-319-48992-6{_}17.

[5] *Architectural Decision Records — adr.github.io*. Tech. rep. Date Accessed: 2020-03-05. URL: https://adr.github.io/.

[6] A. Arela. *Päätöksentekoon vaadittavan tiedon koostaminen viranomaispäätösjärjestelmässä*. Tampere, July 2019. URL: http://urn.fi/URN:NBN:fi:tuni-201906142021.

[7] K. Beck, M. Beedle, A. v. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. *Manifesto for Agile Software Development*. 2001. URL: http://agilemanifesto.org/.

[8] G. Booch. "The accidental architecture". In: *IEEE Software* 23.3 (May 2006), pp. 9–11. ISSN: 07407459. DOI: 10.1109/MS.2006.86.

[9] J. Bosch. “Software architecture: The next step”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3047 (2004), pp. 194–199. ISSN: 03029743. DOI: 10.1007/978-3-540-24769-2{_}14. URL: http://link.springer.com/10.1007/978-3-540-24769-2_14.

[10] Business Finland. *Business Finland*. URL: https://www.businessfinland.fi/.

[11] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford. “Documenting software architectures: views and beyond”. In: *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 2003, pp. 740–741. ISBN: 0-7695-1877-X. DOI: 10.1109/ICSE.2003.1201264. URL: http://ieeexplore.ieee.org/document/1201264/.

[12] B. De Martino, D. Kumaran, B. Seymour, and R. J. Dolan. “Frames, biases and rational decision-making in the human brain”. In: *Science* 313.5787 (Aug. 2006), pp. 684–687. ISSN: 00368075. DOI: 10.1126/science.1128356.

[13] M. Drury, K. Conboy, and K. Power. “Obstacles to decision making in Agile software development teams”. In: *Journal of Systems and Software* 85.6 (June 2012), pp. 1239–1254. ISSN: 01641212. DOI: 10.1016/j.jss.2012.01.058. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121212000374.

[14] R. Faber. “Architects as service providers”. In: *IEEE Software* 27.2 (Mar. 2010), pp. 33–40. ISSN: 07407459. DOI: 10.1109/MS.2010.37.

[15] G. Fairbanks. *Just Enough Software Architecture, A Risk-Driven Approach*. Marshall & Brainerd, 2010.

[16] B. Foote, J. Yoder, N. Harrison, and H. R. Addison-Wesley. *Pattern Languages of Program Design 4 edited by Big Ball of Mud Contents Big Ball of Mud*. Tech. rep. 1999. URL: http://www.laputan.org/mud/mud.html8/28/01http://www.laputan.org/mud/mud.html8/28/01.

[17] M. Fowler. “Who needs an architect?” In: *IEEE Software* 20.5 (2003), pp. 11–13. ISSN: 07407459. DOI: 10.1109/MS.2003.1231144.

[18] M. Fowler and J. Lewis. *Microservices*. 2014. URL: https://martinfowler.com/articles/microservices.html.

[19] M. Galster, S. Angelov, M. Meesters, and P. Diebold. "A multiple case study on the architect's role in Scrum". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10027 LNCS. Springer Verlag, Nov. 2016, pp. 432–447. ISBN: 9783319490939. DOI: 10.1007/978-3-319-49094-6{_}29.

[20] ISO/IEC/IEEE 42010:2011. *Systems and software engineering — Architecture description*. Tech. rep. International Organization for Standardization, May 2011.

[21] A. Jansen and J. Bosch. "Software architecture as a set of architectural design decisions". In: *Proceedings - 5th Working IEEE/IFIP Conference on Software Architecture, WICSA 2005*. Vol. 2005. IEEE, 2005, pp. 109–120. ISBN: 0769525482. DOI: 10.1109/WICSA.2005.61. URL: http://ieeexplore.ieee.org/document/1620096/.

[22] G. Klein. "Naturalistic decision making". In: *Human Factors* 50.3 (June 2008), pp. 456–460. DOI: 10.1518/001872008X288385. URL: http://www.ncbi.nlm.nih.gov/pubmed/18689053.

[23] L. Lundberg, J. Bosch, D. Häggander, and P.-O. Bengtsson. "Quality attributes in software architecture design". In: *IASTED 3rd International Conference on Software Engineering and Applications*. IASTED/Acta Press, 1999, pp. 353–362.

[24] J. McAvoy and T. Butler. "The role of project management in ineffective decision making within Agile software development projects". In: *European Journal of Information Systems* 18.4 (Aug. 2009), pp. 372–383. ISSN: 0960-085X. DOI: 10.1057/ejis.2009.22. URL: https://www.tandfonline.com/doi/full/10.1057/ejis.2009.22.

[25] V. Mezhuyev, O. M. Lytvyn, I. Pershyna, O. Nechuiviter, O. O. Lytvyn, V. Lavrik, O. Kovalska, and Y. Gunchenko. "Acceptance of the Methods of Decision-making". In: *Proceedings of the 2019 8th International Conference on Software and Computer Applications - ICSCA '19*. Vol. Part F1479. ICSCA '19. New York, New York, USA: ACM Press, 2019, pp. 199–204. ISBN: 9781450365734. DOI: 10.1145/3316615.3316677. URL: http://doi.acm.org/10.1145/3316615.3316677.

[26] N. B. Moe and A. Aurum. "Understanding Decision-Making in Agile Software Development: A Case-study". In: *2008 34th Euromicro Conference Software Engineering and Advanced Applications*. IEEE, Sept. 2008, pp. 216–223. ISBN: 978-

0-7695-3276-9. DOI: 10.1109/SEAA.2008.55. URL: http://ieeexplore.ieee.org/document/4725725/.

[27] R. S. Nickerson. "Confirmation bias: A ubiquitous phenomenon in many guises". In: *Review of General Psychology* 2.2 (June 1998), pp. 175–220. ISSN: 10892680. DOI: 10.1037/1089-2680.2.2.175. URL: http://journals.sagepub.com/doi/10.1037/1089-2680.2.2.175.

[28] D. E. Perry and A. L. Wolf. "Foundations for the study of software architecture". In: *ACM SIGSOFT Software Engineering Notes* 17.4 (Oct. 1992), pp. 40–52. ISSN: 0163-5948. DOI: 10.1145/141874.141884. URL: https://dl.acm.org/doi/10.1145/141874.141884.

[29] M. Poppendieck and T. Poppendieck. *Lean Software Development: An Agile Toolkit: An Agile Toolkit*. Addison-Wesley, 2003.

[30] C. Pretorius. "Beyond Reason: Uniting Intuition and Rationality in Software Architecture Decision Making". In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Mar. 2019, pp. 275–282. ISBN: 978-1-7281-1876-5. DOI: 10.1109/ICSA-C.2019.00056. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85066461547&doi=10.1109%2FICSA-C.2019.00056&partnerID=40&md5=446cd276ff25afe3d2aee30977e2476c.

[31] S. Ramakrishnan. "On Integrating Architecture Design into Engineering Agile Software Systems". In: *Issues in Informing Science and Information Technology* 7 (2010), pp. 9–25.

[32] W. W. Royce. "Managing the development of large software systems: concepts and techniques". In: *9th international conference on Software Engineering*. 1987, pp. 328–338.

[33] P. Runeson and M. Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (Apr. 2009), pp. 131–164. ISSN: 1382-3256. DOI: 10.1007/s10664-008-9102-8. URL: http://link.springer.com/10.1007/s10664-008-9102-8.

[34] K. Schwaber and M. Beedle. *Agile software development with Scrum*. Prentice Hall Upper Saddle River, 2002.

[35] M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, 1996.

[36] K. Shumaiev, M. Bhat, O. Klymenko, A. Biesdorf, U. Hohenstein, and F. Matthes. "Uncertainty Expressions in Software Architecture Group Decision Making: Explorative Study". In: *Proceedings of the 12th European Conference on Software Architecture Companion Proceedings - ECSA '18*. ECSA '18. Univ Politecnica Madrid, Campus Excelencia Internac; Univ Politecnica Madrid, Escuela Tecnica Super Ingn Sistemas Informaticos. New York, New York, USA: ACM, 2018, pp. 1–8. ISBN: 9781450364836. DOI: 10.1145/3241403.3241447. URL: http://dl.acm.org/citation.cfm?doid=3241403.3241447.

[37] H. A. Simon. "Satisficing". In: *The new Palgrave: a dictionary of economics* 4 (1987), pp. 243–245.

[38] A. Tang and H. van Vliet. "Software Designers Satisfice". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9278. 2015, pp. 105–120. DOI: 10.1007/978-3-319-23727-5{_}9. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84975744369&doi=10.1007%2F978-3-319-23727-5_9&partnerID=40&md5=3eeab5d63d8b0e7eaaf7f137e8fa0fb0.

[39] A. Tversky and D. Kahneman. "The framing of decisions and the psychology of choice". In: *Science* 211.4481 (Jan. 1981), pp. 453–458. ISSN: 00368075. DOI: 10.1126/science.7455683. URL: http://www.sciencemag.org/cgi/doi/10.1126/science.7455683.

[40] H. van Vliet and A. Tang. "Decision making in software architecture". In: *Journal of Systems and Software* 117 (July 2016), pp. 638–644. DOI: 10.1016/j.jss.2016.01.017. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84956623197&doi=10.1016%2Fj.jss.2016.01.017&partnerID=40&md5=8a06ea5ab015ff3e0576015892d0d3c9.

[41] A. Zalewski, K. Borowa, and A. Ratkowski. "On Cognitive Biases in Architecture Decision Making". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10475 LNCS. 2017, pp. 123–137. DOI: 10.1007/978-3-319-65831-5{_}9. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029001329&doi=10.1007%2F978-3-319-65831-5_9&partnerID=40&md5=f229eca15bcc151e2031298fe2e5ddde.

[42] Y. Zhang, R. K. E. Bellamy, and W. A. Kellogg. “Designing Information for Remediating Cognitive Biases in Decision-Making”. In: *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 2015, pp. 2211–2220. ISBN: 9781450331456. DOI: 10.1145/2702123.2702239. URL: http://dx.doi.org/10.1145/2702123.2702239.