# Agile software development and IT-architecture interactions in the public sector

A multi-case study approach to identify whether these roles are complementary or counterproductive.

Master thesis submitted to Delft University of Technology

in partial fulfilment of the requirements for the degree of

**MASTER OF SCIENCE**

in **Complex Systems Engineering & Management**

Faculty of Technology, Policy and Management

by

Constantijn van der Vliet

Student number: 4468767

To be defended in public on 07-07-2022

## Graduation committee

| | |
|---|---|
| Chairperson | : Prof.dr.ir. M.F.W.H.A. Janssen, Section ICT |
| First Supervisor | : Prof.dr.ir. M.F.W.H.A. Janssen, Section ICT |
| Second Supervisor | : Dr. H.G. van der Voort, Organisation & Governance |
| External Supervisor | : Drs. W.G.P. Heijnen, Digital Enablement KPMG |

# Preface

I managed to gather very interesting case studies by I speaking to some very interesting individuals and was able to compare different perspectives on software development due to that. Consequently, I would like to thank my supervisors and colleagues for their sharing their insights and opening doors to these people. I would like to thank all interviewees for participating and hope you will find use in this thesis in current and future software development processes. Finally, I would like to express my thanks for the support from family and friends.

# Summary

The interaction between two software roles was investigated in the public sector: Agile software development and IT-architecture. The objectives of this study were to explore and define a typology of interaction models, provide a description of how these models are affected by various context factors, provide descriptions of complementary added value and problems, as well as a set and explanation of general governance strategies that can help to obtain complementary added value from agile software development and software architecture. Another objective was to provide practitioners with a new perspective on the relevance of other roles in software development and to use this new perspective in software development processes to obtain higher quality software while using less resources. Finding governance strategies that obtain complementary[1] added value from architecture-agility interactions could help organisations to improve the quality of their software products and to prevent negative societal consequences of quality issues in public sector software. Second, it could also help to reduce software development and maintenance costs. Moreover, timely delivery of quality software could enable an organisation to execute its business[2] processes faster and cheaper.

In a software development process issues can arise with scope, resources and time. Either resources and time are overrun or the product that has been developed does not meet the scope, meaning the requirements of the client and end-users are not met. Issues often arose from a gap in understanding between client, end-users and developers, but also from integration of the components later in a plan-driven waterfall development approach. To cope with unsatisfactory results upon delivery to the client, but also with integration phase issues and the pressure big deadlines in high uncertainty environments the agile methodology was designed. The agile methodology reckons with the plan-based, phase gated[3] waterfall approach and introduces a short-cyclical iterative approach. Developers work in sprints, slice and timebox work expressed in user stories that have been refined, involve all relevant stakeholders, prioritise these stories only a couple of sprints ahead, do a retrospective at the end of the sprint and daily stand-ups to identify issues early and re-evaluate the sprint planning. As software that could be put into production is delivered at the end of each sprint, this approach creates a predictable 'heartbeat' for the client and allows developer and the customer to decide together what is most important and which work should be committed to by the development team. Reducing issues that arise if business commits without involving IT and providing relevant stakeholders with the opportunity to pose their criticism to what has been built at the end of each sprint instead of at phase gates only.

Still, not everyone is convinced that the agile methodology is 'the way' to develop software. Critics state that oversight can be lost, inconsiderate attention is given to issues which are not directly visible, and that this methodology is prone to losing long-term perspective. In order to reduce risk, big upfront development is used to identify and plan for these risks. This latter role is filled by IT-architects, which have an advisory role to design and safeguard the current and future business and IT landscape. They do so by setting constraints for developers, focusing on quality attributes rather than new functionality, making plans, designs and communicating those with all relevant stakeholders. In theory, this creates misalignment or even conflict between the roles of architects and software developers, but could they also foster co-creation from with the organisation can gain added value? How can managers and leaders organise their organisation in such a way that both the long-term and short-term perspectives are considered?

In order to answer these questions multiple organisations have been investigated through a multi-case study approach. Ten practitioners from six organisations have been interviewed to reflect on a specific software development process in the public sector. Two practitioners from the private sector were added as a validation case study. Interviews lasted 1,5-2 hours each and were transcribed by hand

[1] The sum of the parts is greater than the sum of individual parts alone.
[2] Business as in core business, so this term can also be used in a government context. For example, a business process of a municipality could be to provide citizens with a new passport.
[3] Start with a planning phase, then move on to the development phase, then testing and integration phase etc.

due to the sensitivity of the contents. Where possible one role from a software development perspective and one role from an architectural perspective has been interviewed. Which resulted in interviews with:

- 2 development team leaders/solution architects[4],
- 2 back-end developers,
- 3 enterprise architects,
- 1 domain architect,
- 1 Scrum master/functional designer,
- 1 product owner,
- 1 senior product manager, and
- 1 domain lead.

These perspectives were bundled together in case studies in order to identify similarities and differences in statements from practitioners. Analysis has been done by colour coding transcripts and organising statements into tabular displays. Then the case studies were analysed cross-case to identify differences and similarities between interaction models and governance strategies in order to distil good and bad practices.

The importance of communication and knowledge as context factors of the interaction from literature have been validated. Trust, stability and perceptions of both role were found to influence the case studies as well across several case studies. Trust, stability and perceptions of both roles were new contributions to literature. The influence of uncertainty and risk as context factors to determine a balance point for agile-architecture interactions by Waterman (2018a & 2018b) has been validated. Risk and uncertainty played a role on three different aspects: 1) Requirements, 2) Technology and 3) Staffing. All context factors were able to influence the interaction of IT-architecture and agile software development in both positive and negative ways. Based on the influences of the context factors that were identified and the differences and similarities of balanced exchange interaction models, a first iteration of a typology has been expanded. The balanced exchange interaction model has been split up into four new exchange interaction models, resulting in a total of six possible interaction models:

1) It-Architecture dominant interaction model;
2) The carry over or ping-pong model;
3) The louse in pelt model;
4) The solution architect as cooperating foreman;
5) The co-development model; and
6) Agile dominant interaction model.

Contributing to academic literature and practitioners knowledge with two very extreme ends and four reference points for balanced exchange models. Tensions and bottlenecks that have been identified in case studies can shed a light on what problems were found in the interaction models that organisations used. This information adds to academic literature, as it gives substance to the typology. This information could be used by practitioners to identify the associated tensions and bottlenecks for their own interaction model. Recurring bottlenecks across interaction models were:

- Hiring of staffing with the right knowledge.
- Scalability of agile dominant exchange models, coordination issues arose on larger scales.
- Lack of formalisation or recognition of roles.

Reoccurring tensions across interaction models were:

- Short- versus long-term perspectives in combination models.

[4] Double role

- The IT-architecture should enable agile software development.
- Agile and the government context.

It was found that every interaction model provided added value, except for the IT-architecture dominant model. For this model added value was not discussed by participants. The main complementary added value of the combination was the ability to balance up-front design with the agile process to address roadblocks or issues. The agile-architecture interactions complementary was found in alleviating the problems that occurred when IT-architects and agile developers worked side-by-side and did not communicate effectively. Which led to either problems with sustainability of the solution, as quality attributes were not addressed or to problems with functionality, as the wrong thing had been built, since it has never been shown to an end-user. Governance strategies could be used to obtain complementary added value in agile-architecture software interactions by:

- Coping with coordination issues in scaling;
- Moving away from directionally composed IT-architectures towards iterative IT-architectures;
- Addressing agile in a government context;
- Coping with a lack of resources or knowledge;
- Addressing the importance of formalisation and recognition;
- Coping with the product owner role;
- Balancing up-front and agile architecture; and
- Coping with risk and uncertainty.

Practitioners need to invest in training of both roles in the processes and ideas of the roles they interact with. This would help both roles to understand each other better and to alleviate tension. This could help them to become more mature in both IT-architecture and agile software development as an organisation. Since there is scarcity in knowledgeable IT-resources, especially for solution architects, there is an opportunity for the government to fill this gap by investing in good IT education. Future research is needed to see whether the typology holds in different types of organisations and to identify whether the governance strategies can be replicated.

# Contents

# 1. Introduction

## 1.1. Research problem

A study from 2012 by the University of Oxford and McKinsey stated that 66% of large software projects overrun costs, 33% experiences schedule overruns and in 17% benefits fall short of expectations (Bloch et al., 2012). A crucial element to deliver expected value was found to be effective teams and alignment of their incentives with overall goals of the project.

This research addresses the alignment of two different roles in software development that are often combined in software development projects: Agile developers and IT-architects. *"Agile developers are building before the outcome of the product is fully understood, adjust plans and designs as empirical knowledge is gained while building, trust the judgement of those closest to the problem and encourage continuous collaboration with the customers"* (Madison, 2010, p. 1). While IT-architects create design patterns, enhance quality attributes,[5] establish a technology stack[6] and communicate with all stakeholders (Madison, 2010). IT-architects use up-front planning and design to establish functional and non-functional requirements to create a sustainable IT-landscape that matches with the business[7] landscape and desired performance (in terms of security, scalability, interoperability etc.) (Bellomo et al., 2015; Madison, 2010; Waterman, 2018a, 2018b; Woods, 2015). In contrast, agile software developers use short iterations in consultation with the product owner[8] and end-users to respond to changes, capture feedback early and to create business value early in the development process. (Beck et al., 2001a). The latter approach tends to bias allocation of development time and resources to functional requirements, as these directly add value to the customer in terms of new or improved functionality (Waterman, 2018a). Consequently, agile software development can result in problems like extra development time and costs later in the design process, for example with security, maintainability, or interoperability, as these quality attributes receive less attention than functional requirements that deliver visible business value (Bellomo et al., 2015; Madison, 2010). Up-front architecting can reduce the adaptability of the design, incur extra costs by having to change designs once customer requirements are better understood and incur costs due to redundant work which will not be featured in the end-product (Waterman, 2018a; Yang et al., 2016). In short, architecture specialists tend to live in a more rigid, 'paper-world,' while agile software development is aimed at adaptability to deal with the unruliness of practical implementation (Pers. Comm., Janssen, 2021). Both practices approach a similar goal with best intentions, but in opposite ways which can result in clashes, as it is easy for both sides of experts to blame the issues on the opposite party. Hence a shared understanding on issues in the software development process is not evident, but identified problems hint at a need for cooperation and balance of these long- and short-term perspectives. Researchers have already identified that successful interaction, or even combination, of both roles does not reside in theoretical issues, but in practical matters of adoption (Falessi et al., 2010; Poort, 2014; Woods, 2015). Thus, while the principles and values of both roles and their practices can be interpreted as clashing, it would be interesting to identify how they could be complementary in practice through an empirical study.

[5] Often referred to as non-functional requirements.
[6] Collection of programming languages, frameworks, database, front- and back-end tools, API's etc. that form the combination of technologies that an organisation uses to build and run applications.
[7] The organisational processes that create revenue in the private sector or the public service provision in the public sector.
[8] Often the client or appointed by the client.

## 1.2. Research objective

The objectives of this study are to explore and define a typology of interaction models, provide a description of how these models are affected by various context factors, provide descriptions of complementary added value and problems, as well as a set and explanation of general governance strategies that can help to obtain complementary added value from agile software development and software architecture. Consequently, the aim of the research is to provide results that create a new perspectives on the roles of IT-architecture and agile software developers in software development that future research can expand upon.

Another objective is to provide practitioners with a new perspective on the relevance of other roles in software development and to use this new perspective in software development processes to obtain higher quality software while using less resources. Finding governance strategies that obtain complementary[9] added value from architecture-agility interactions can help organisations to improve the quality of their software products and to prevent negative societal consequences of quality issues in public sector software. Second, it could also help to reduce software development and maintenance costs. Moreover, timely delivery of quality software could enable an organisation to execute its business[10] processes faster and cheaper.

Complementarity is an important concept to understand the objective of this research, as it means that a combination of multiple things leads to an added value that is greater than the sum of the individual things alone. For example, for many the combination of red wine and a steak is better enjoyed in combination than the enjoyment of individual red wine and a piece of red meat summed up. This phenomenon could also work in a negative way, by making things worse in combination than they would have been individually, for example white wine and a piece of red meat in combination would provide less joy to many, as it reduces the enjoyment of both the wine and the meat, while individually they would have been more enjoyable. This is called counterproductivity in this thesis. Thus, complementarity is important as this allows practitioners to reap benefits from the combination or interactions that are greater than the benefits of side-by-side implementations agile software development or IT-architecture. Consequently, the identification of complementary added value would imply that the added value for practitioners can only be obtained by practising agile software development and IT-architecture in a combination.

[9] The sum of the parts is greater than the sum of individual parts alone.

[10] Business as in core business, so this term can also be used in a government context. For example, a business process of a municipality could be to provide citizens with a new passport.

## 1.3. Suitability for Complex Systems Engineering & Management programme

This thesis is part of obtaining a degree in the CoSEM master programme. The CoSEM master programme focusses on the complexity of engineering in socio-technical systems. Software is a clear example of a socio-technical system, as it is a technical system that is built to support or automate human processes. These processes are often public service provision in the public sector. Consequently, how these technical systems are designed, build and implemented have an effect on human lives and society at large, especially in the public domain (ANP & Doorenbosch, 2019; Parlementaire ondervragingscommissie Kinderopvangtoeslag, 2020, pp. 7-9 & 13-15).

While an answer to how these distinct roles could be complementary is deceivingly straightforward, determining how architecture and agility lead to added value or problems that affect an organisation is complex in itself, without complementarity (Falessi et al., 2010; Poort, 2014; Woods, 2015). The object of study is the design and engineering processes of software development, there are different perspectives on how to approach these processes. These perspectives can cause people in one role to seek or avoid interaction with the other role. The research focusses on the governance of social interactions, or process management, of engineering approaches for software technology and goes a step further by identifying whether these interactions are of a complementary nature. Identifying this complementary nature of social practices around a technical engineering approach with a large societal impact seamlessly fit in the CoSEM programme.

Moreover, software development processes in the public sector involves a myriad of stakeholders from different organisations and with both distinct and overlapping roles and responsibilities which affect software development processes. To further add to complexity, each organisation will have assigned and defined roles and responsibilities differently. The same could be said on what agile software development is and how it is employed in an organisation, or how to define the role of any type of IT-architect[11] and what their tasks and responsibilities are in an organisation. Identification and analysis of these problems requires a systemic and creative approach, in this case a multi-case study approach, to assess the impact interaction in these technical software development processes on the organisations that are studied. Thus, the societal impact, the socio-technical nature, myriad of stakeholders and perspectives involved and additional complexity added by complementary outcomes make this research a suitable graduation thesis subject for the Complex Systems Engineering and Management master programme.

## 1.4. Structure of thesis

Section 2 will provide knowledge gaps, a main research question and sub-questions by means of a literature review. Section 3 will discuss the research approach and methods, followed by, explanation on which data needs to be collected and tools used to conduct the research. Section 4 will present how agile-architecture interactions work according to academic and grey literature. Section 5 will discuss how the case studies and interviewees were selected and classify the case studies on a basic typology. Section 6 will discuss how the case studies were influenced by context factors, devise an extension to the basic typology and reclassify the case studies using this extended typology. Section 7 will describe how interaction models of the extended typology were affected by bottleneck and tensions. Section 8 does the same for added value. Section 9 will discuss governance strategies that help to obtain complementary added value. Section 10 will conclude these findings and discuss the limitations, as well as scientific and practical implications of the study. This section will also provide recommendations for further research.

[11] There are multiple types of architects, some examples: chain, enterprise, domain, technical, business, application, cloud, data, solution, system and software architect.

# 2. Background

In this section core concepts are defined, and the main findings of a literature review are discussed. Moreover, multiple research gaps will be analysed in order to define a research question.

## 2.1 Definition of core concepts

### 2.1.1 Agile and agility

According to Cao et al. (2009) agile is an iterative software development process that uses frequent consultation with the customer, small and frequent releases, and rigorously tested code. According to them it is often implemented by organisations in order to become more competitive, improve processes and reduce costs. It can therefore be stated that agile development is focused on achieving agility, which is the ability to adapt to changes quickly (Gong, 2012). Agile is a more lightweight methodology compared to traditional software development methods, such as waterfall, the V-model and RUP, which are more plan-based sets of sequential steps like requirements elicitation, solution design, solution development, testing and deployment[12] (Loo et al., 2012). Thus, traditional methodologies require a stable and documented set of requirements as based on this the architectural designs are made, software is built and tested. Requiring large rework phases if requirements change or new requirements become apparent after the requirements elicitation process in traditional methods. However, it does allow to determine costs, schedule and allocate resources accordingly. Figure 1 shows how in agile, the agile software development lifecycle is repeated iteratively to build a software product in small steps and cumulatively build value through iterations of the software development lifecycle as opposed the waterfall model with a large outcome at the end (Harris, 2021). The waterfall model is used is this example, but the idea is the same with other traditional models such as the V-model: big value delivery at the end versus cumulative value delivery through iterations.

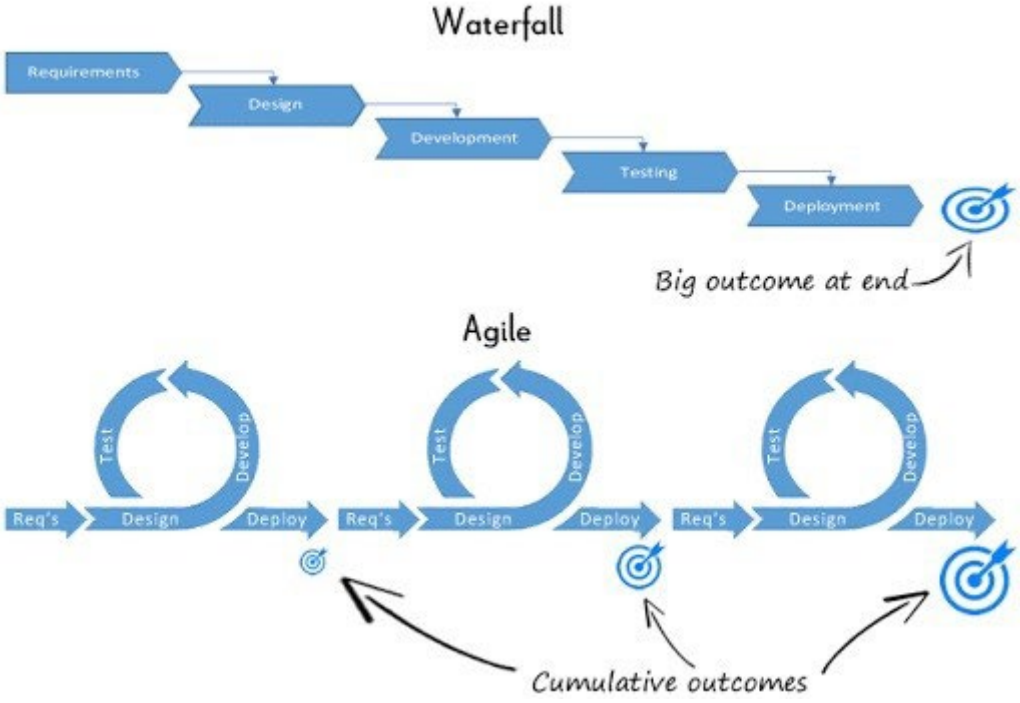


*Figure 1: Waterfall vs Agile from Harris (2021).*

[12] Different models use different steps and sequences of those steps.

The role of an agile software developer is to create a software product through code and to communicate with the customer (end-user) and product owner on how they perceive the code-based product through short-cyclical iterations. The product owner is (a representative of) the client who is responsible for the outcome of the software development process. The product owner is the primary contact for stakeholders, such as end-users. Agile is used to improve the communication between the product owner, end-users, and the developers of software, as information asymmetry can be very large between the (non-technical) product owner and software development team. For example, product owners often have difficulty in explaining requirements, especially before the development process has started. Likewise, it can be difficult to explain software related capabilities and constraints to product owners. Consequently, this creates uncertainty in what the client wants and how the clients' needs should be addressed resulting in changes to requirements and new requirements that become apparent after the requirement elicitation process. Agile development is a way of coping with this uncertainty in the realisation of software solutions by going through the software development lifecycle for a piece of the software and presenting this to relevant stakeholders. Allowing to capture feedback earlier in the development process.

The agile manifesto presents four values that underline the agile working practices (Beck et al., 2001a, p.1)[13]:

1) *"Software that works over comprehensive documentation,*
2) *Customer collaboration over contract negotiation,*
3) *Individuals and interactions over processes and tools,*
4) *Response to change over following a plan."*

These values and principles are currently put into practice in various ways. For example, by collaborating with customers to create user stories, which are semi-structured ways of phrasing (functional) requirements in an informal, descriptive way from the perception of the customer. This helps to address the communication gap between developers and customers. User stories are refined with relevant stakeholders to ensure that developers have enough information to time box and implement them correctly. A Definition of Ready (DoR) can be used to define when a user story is ready. These user stories are then time boxed by the team to estimate how much work the implementation of this user story is and put on the backlog. The backlog makes transparent which user stories still must be done, similarly, there is an overview of which stories are under development, review, being tested and which stories are finished. Sprints are used to deliver a potentially shippable product increment, usually in a short and set time. This sprint increment could be an improvement or change to a piece of running software, as well as wholly new release. During sprint planning events user stories are prioritised by the product owner (PO) and development team. Since the user stories are time boxed and the amount of time the developers have available in the next sprint is known, the work can be committed to and planned by the developers. This means that the deliverable at the end of a sprint is a piece of software that is coded, tested, integrated and usable at the end of the sprint to deliver business value.[14] This deliverable is often called a MVP, minimum viable product, as this is the minimum product is needed to attract early end-users and capture feedback on the product early in the development life cycle. This deliverable is assessed based on a Definition of Done (DoD). Since work is time-boxed and available time and resources are known, work can be prioritised through the backlog. Thus, the scope of the project is flexible and the team is more able to cope with new or changing requirements.

[13] These values have been applied in several frameworks such as LeSS, SAFe, Scrum and Spotify.

[14] I.e. increased process efficiency, full process automation, enabling a new business process.

Scrum is a framework to encourage the self-organisation of agile development teams and describes how to perform the previously described processes and others. It uses a backlog and sprint planning's to divide work into sizable chunks and prioritise this work for small development teams on a small scale.[15] Other frameworks, all with a distinct flavour or context in which they are useful are Spotify, LeSS and SAFe. These frameworks are based on Scrum but focus integration of agile on a larger part of the organisation and scaling of agile teams for example. These frameworks create a shared understanding and language as well as governance strategies that define roles, tasks, responsibilities, monitoring and control elements in the organisation and software development process. Agile fanatics criticise the idea of steering or managing through processes, documentation and planning in agile software development based on the values of the agile manifesto, as agile can be seen as bottom-up approach to software development, driven by decentralised, multi-disciplinary and self-organising teams.

According to Gong (2012) agility includes the ability to respond to unpredictable changes and the ability to rapidly reconfigure to a new parameter set from an operations management perspective. A way of implementing Business Process Management (BPM) is IT-architecture. From a BPM perspective, agility can be expressed as flexibility and the amount of speed in modifying or reconfiguring business processes. Flexibility can then be defined as the ability to respond to predictable changes based on the existing configuration of parameters that were pre-established from an operations management perspective. From a BPM perspective flexibility can be defined as the ability to deal with both unforeseen and foreseen changes, maintaining effectiveness, and limiting the impact of changes (in other parts).

### 2.1.2 Software and Enterprise architecture

There are various architecture roles in IT. According to The Open Group Architecture Framework (TOGAF), the enterprise architect is a role that translates the enterprise[16] vision and strategy into requirements and principles. These could be business, information, application or hardware related and for each of these elements of the enterprise separate architecture roles can exist. The requirements and principles could also cut across various of these abstractions of organisational layers compartments.[17] The enterprise architect then coordinates these other IT-architecture roles, such as business architects or application architects, as it the responsibility of the enterprise architect to create a sustainable IT-landscape that fits with the business landscape. Thus, the enterprise architect can be said to be closest to the most recognisable governance processes and decisions from organisational management. The enterprise architect does not define answers to how questions or what these requirements or principles mean for the solution(s).[18] Different IT-architecture roles do so, commonly referred to by the related role, i.e. a software architect translates the work of the enterprise architect into a technical and sustainable piece of software, which is then developed by engineers/developers. In another example, the business architect/analyst will seek for alignment with the business processes and so on. All architecture roles design an IT-architecture and document this IT-architecture, which can be done in various ways from word-documents to wikis. Moreover, they design tests in order to assess whether the system is built as it was designed. Thus, they define monitoring and control mechanisms, which is a form of governance to ensure compliance to the IT-architecture designs. This design process is done up-front, so before developers start developing and follows a traditional waterfall approach.

[15] Up to 3-5 teams.
[16] organisation
[17] Often recognised in compartments, silos or layers such as in TOGAF.
[18] A software application and it's supporting processes and hardware are often referred to as a solution.

Similar added value to that of agile software development (among other types of added value) is found for enterprise architecting in research by Gong and Janssen (2019). Enterprise architecture is defined by Lankhorst (2009 , p. 3) as *"a coherent whole of principles, methods and models that are used in the design and realisation of an enterprise's organisational structure, business processes, information systems, and infrastructure."*[19] This thus concerns enterprise-wide systems and can be hardware and software related, which need to be aligned with business values. The enterprise architect both looks at the current IT-architecture, while also gathering requirements for the future IT-architecture of the organisation (Hanschke et al., 2015). Research by both Lankhorst (2009) and Gong & Janssen (2019, p. 4;2020, p. 1) claim that enterprise architecture can help to achieve agility.

Yang et al. (2016) did a systematic mapping study of Software Architecture (SA) and agile software development. The definition of software architecture differs from the definition of enterprise architecture: "*the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*" (ISO, 2011, as cited in Yang et al., 2016, p.1). One can see here that the focus has shifted from the entire enterprise and business to a single software system. However, the environment and relationships with this environment are mentioned as well. Therefore, I argue that there is overlap between the two definitions, as a software system is never really a standalone element and is part of or functioning in at least one enterprise environment. The software architecture or information systems architecture[20] builds on components of the technology architecture (The Open Group, 2018). In turn business services run on the applications from software architecture. All these (Enterprise) architecture components are influenced by architectural governance.[21] Moreover, two or more software systems (within the same layer) might need to be able to interact as well (Bellomo et al., 2015). Thus, in this study I will treat both enterprise and software architecture as IT-architecture practices, but with a different scope: the former on enterprise system level, the latter on software system level. Thus, the designs of the software architect are more granular and based on the less granular designs of the enterprise architect.

IT-architecture is a more top-down approach to software development, where the enterprise architects define the enterprise architecture, and other IT-architects have to fill in their own IT-architecture in such a way that it adheres to the enterprise architecture. So software architecture is mainly concerned with the interactions of the software system that is designed to the enterprise environment[22], which explains the emphasis of the software architecture role on non-functional requirements, design principles, design decisions, components and documentation. Consequently, software architecture involvement is most prominent after the requirements are well understood and designs of the software architecture are needed in the waterfall software development lifecycle, as depicted in figure 2 (Mihaylov, 2015c & 2015d).

[19] Note this is one of many definitions, as there are hundreds to be found.
[20] Information systems architecture is what comes closest to software architecture in TOGAF.
[21] Architectural realisation in TOGAF, consisting of: Implementation Governance, Opportunities, Solutions & Migration planning.
[22] Other elements in the information systems/application layer, but also elements in other layers, i.e. technology and business layers.

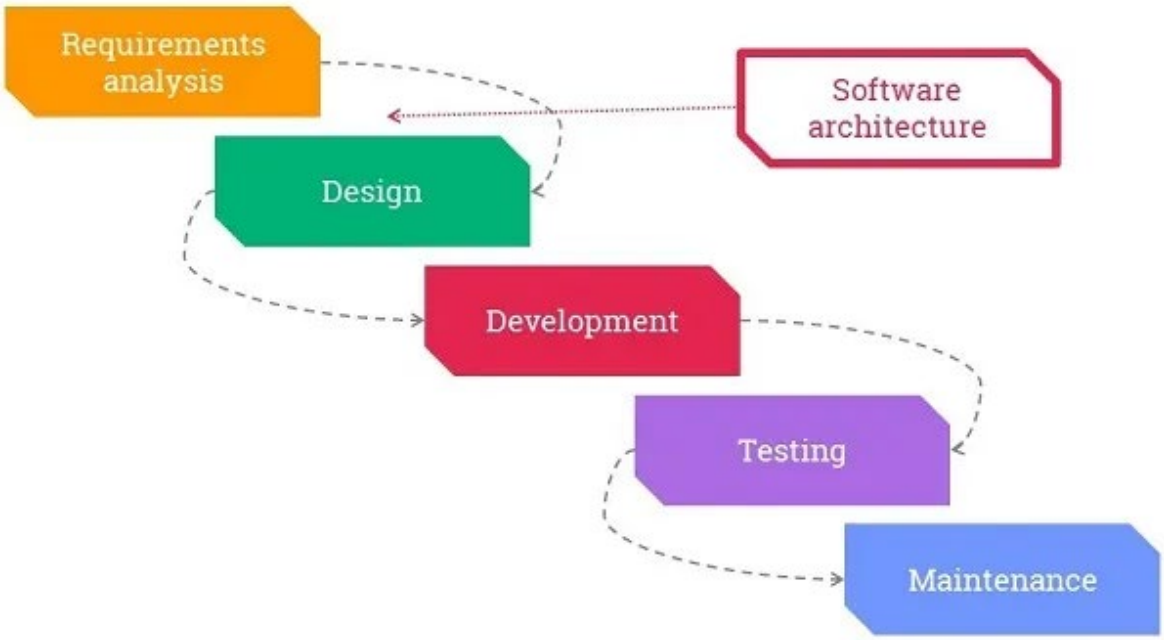


*Figure 2: Software architecture in traditional waterfall development (Mihaylov, 2015d)*

A traditional software architect is usually is a person with strong technical knowledge and experience, often a promoted developer (Mihaylov, 2015c). The traditional software architect usually has four features:

1) Focus on the big picture, as there is a need to consider the current and future landscape.
2) Produces blueprints, documents and diagrams describing the software architecture from different perspectives that enable developers to develop a system.
3) Not much hands-on experience, as they are rarely hands-on involved in the development process and generally moves on to another project when the designs are finished, leaving the developers with the designs.
4) Compliance-oriented, such as legislative norms, standards, licences etc.

Both enterprise and software architecture roles need strong communication skills to convey their designs and to be successful in reaching agreement on requirements with various stakeholders. However, it is said that architects fail to communicate in these regards, fail to see operational issues and stick to their paper tigers instead of designing for reality. Both enterprise and software architects are involved in constraint setting, for example a software architect could make a choice to use a specific programming language for a software application. Likewise, an enterprise architect might make a decision to use a (specific) package or decide which components will be used for software that is developed in-house. Because of this, they can be seen as burdensome by both agile and traditional software development teams.

Interestingly, Madison (2010) has found that incremental use of enterprise architecture can help to integrate software architecture and agile software development. IT-architects employ up-front design and documentation to define architecture, principles, vision, requirements, components and standards. These provide a frame and guidelines for the developers which determines priorities and possible solutions. There are various frameworks[23] for IT-architects to do so, for this research, the most interesting are TOGAF and NORA. NORA is a reference architecture for the Dutch government and is aimed at creating a shared understanding and language, defining core concepts (ICTU, 2021a). Moreover, the NORA defines binding standards to assure interoperability and quality service provision

[23] Note that currently, we are talking about IT-architecture frameworks that help to deliver frames for software developers.

by government organisations and thus is used by many, if not all, government organisations (ICTU, 2022). TOGAF mainly concerns enterprise architecture, software architecture can be identified as part of the application[24] or information systems architecture (The Open Group, n.d., 2018,) within the enterprise context.

### 2.1.3 Theoretical framework for governance and governance strategies

Governance is a concept that is hard to define due to the plurality of definitions in literature (Fukuyama, 2016). The term stems from political science, but has spread to almost every thinkable domain, including software development. Since many scientific papers focus on employing a specific part (i.e., requirement elicitation) or type of added value (i.e., security) of software development, a more general definition is needed for this study.

An example of such a more general definition is Software Development Governance (SDG) as defined by Chulani et al. (2008, p.3) as: *"Establishing chains of responsibility, authority and communication to empower people within a software development organization"* and *"Establishing measurement and control mechanisms to enable software developers, project managers and others within a software development organization to carry out their roles and responsibilities."* They state that the objective of software development governance is to make sure that the results of a software organisations business processes meet the strategic requirements of the organisation.[25] Chulani et al. (2008) their definitions of software development governance can be operationalised using three constructs governance, management structure and processes. These constructs are used as a theoretical framework to structure statements by participants on governance.

#### Governance

The governance construct is aimed at steering and determines responsibilities, authority and tasks. Governance determines management structure which entails who has ownership over what and who sets priorities. This includes assignment of the time and resources that are spent on the development of the software product under development. Moreover, governance affects communication through the organisational structure, as it also includes who reports to who and how people report to each other (i.e., hierarchy or network structure), which is important, as poor governance or misalignment of reporting lines can hamper the cooperation in a software development project through conflicting interests (Bloch et al., 2012). Consequently, this can lead to increased costs, development time or reduced customer satisfaction upon delivery of a software product. Thus, being able to identify and address these issues can provide added value.

To operationalise governance further, it is useful to identify that governance has influences on various elements of the organisation. Organisational governance is focused on the whole enterprise, such as who is responsible for which department. While operational governance affects the development process of software directly, for example through a decision on which agile framework is used to assign roles in software development. According to Chulani et al. (2008) software development governance has 3 main concerns: 1) Managing value, aligning software and business on organisational and project levels, balance risk and return and provide clarity and accountability; 2) Develop flexibility, leverage global resources by enabling agile development choices and the use of iterative processes to reduce risk and; 3) Control risk and change, continuously measure to reduce risk, enable lifecycle change management and meet internal and external compliance needs. From this notion, it seems that these authors are more on the agile side of the combination. Governance can steer an organisation to be more suitable or dominant for either agile software development or software architecture (i.e., based on their organisational priorities or authorities) (Waterman, 2018a). We will discuss the concepts of flexible development, risk and change management later in the literature review, as they are identified by other

[24] Modelling languages like ArchiMate and frameworks for architecture use layers to structure thinking about and discussing architecture concepts, the application layer is the most associated to software development. These layers are not 'empirical realities' but conceptual notions.
[25] Or enterprise

authors as well (i.e., Waterman (2018a, 2018b)). Interestingly, the paper by Chulani et al. (2008) identifies that defining an information architecture is important to develop flexibly as a software development governance concern, confirming that the architecture-agility combination is a concern in software development governance, as IT-architecture is needed to develop flexibility in software development.

All development organisations make decisions and have some form of governance, implicit or explicit. Moreover, governance styles vary in different organisations, depending on the fundamental goal of the organisation. While innovative software businesses working on cutting edge solutions might want to be risk-seeking, very agile and only have a few holistic or light weight (governance) processes, business where the software needs to meet high security or quality standards[26] (i.e., hospitals, municipalities etc.) employ more stringent processes (Chulani et al., 2008). Public organisations are more likely to be the latter, as they often hold sensitive personal data and provide critical public infrastructure services. Large organisations are typically more concerned with security, scalability and interoperability as well, since they are more likely to attract regulatory attention. Therefore, large public organisations require more governance to determine what is allowed and what is not to safeguard these requirements.

### Management structure

Managers make decisions about other individuals, such as hiring, firing and salary decisions. Managers also make decisions about human (priorities, time, budget and staffing) and other resources like tools, servers etc. Decision rights of managers are determined by a governance process. Similarly, other individuals might have decision rights assigned to them in order to achieve a designated goal and are monitored by managers. So, where governance assigns decision rights to roles in an organisation (establishing a measurement and control strategy), management is concerned with actually making decisions (execution of decision rights) or monitoring the decisions made by others. Management is also held accountable for decision making and monitoring of others.

### Processes

Managers use or implement processes that developers follow to get work done in order to achieve results. Chulani et al. (2008, p.4) define a process as *"A process is a naturally occurring or designed sequence of operations or events that produces some outcome, possibly taking up time, space, expertise or other resource. In addition, a business process has the rights for certain people to take actions and arrive at decision points to advance the process to the next step. Processes may be characterized by specifying their control points: artifact control point and lifecycle control point. Moving through these control points requires a set of decisions (such as 'phase complete?') which again require associated rights."* The control points in life cycle and artifact may or may not align. For example, the phase gates in a project management approach are lifecycle control points, while artifact control points might be a check whether certain specs or requirements have been implemented. The interactions between governance, processes and management are summarised in Figure 3, this figure is from Chulani et al. (2008, Figure 3, p.5). Be mindful that strategy in this picture concerns enterprise strategy and not governance strategy. Enterprise strategy is presented as a driver for governance. Moreover, communication is missing as this is discussed later, it is worth noting that communication can also be formalised into a process, for example through a verbal progress update, reviews and refinement sessions, presentations etc., but also through documents, such as policy notes, designs, code documentation, or be computer/application mediated, by updating a backlog or Kanban board, updating communication tools such as Slack, Jira, GitHub etc.

[26] Often referred to as compliance requirements, non-functional requirements or quality attributes.

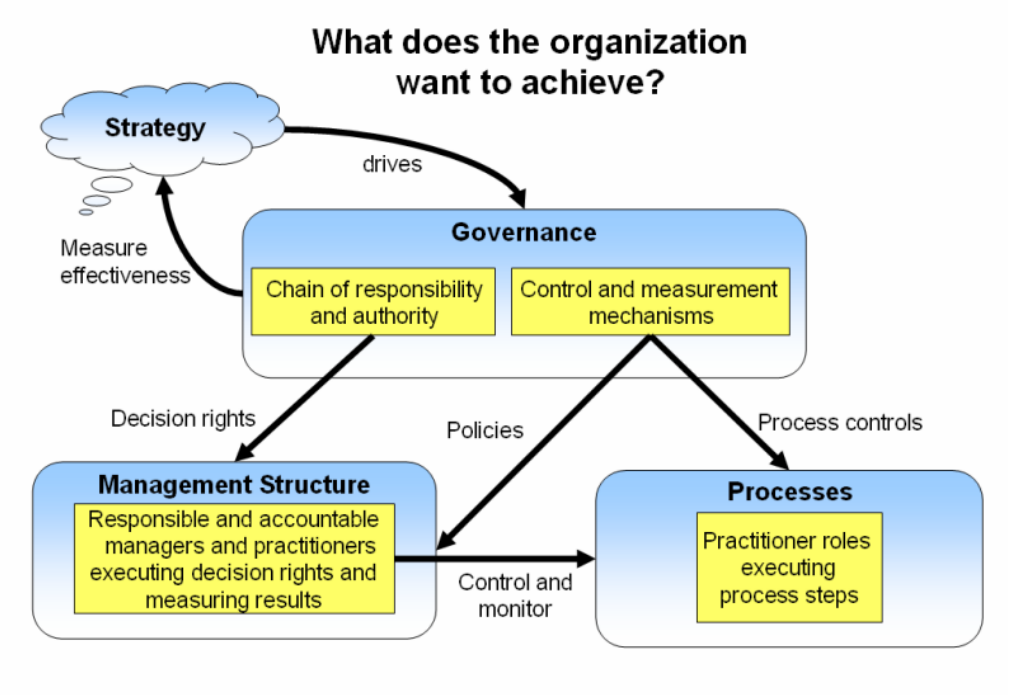


*Figure 3: Governance, Management and Processes from Chulani et al. (2008, Figure 3)*

## Governance strategy

A strategy is a plan of action to achieve a long-term or overall aim. Thus, a governance strategy within the scope of this research is a plan of action to achieve a governance related aim. For example, a governance strategy can be the implementation of a new framework, determining new roles and responsibilities within the development team. Or a change in governance policy that invokes changes in management structure which changes who reports to who and what to report on, (mis)aligning the priorities of the IT-architect with members of the development team. Within governance there is a distinction between formal and informal governance. Where informal are unwritten rules, special favours and reaching understanding outside of the formal system based on rules. Since this is hard to identify, the focus will be on formal governance strategies, which are coded in a rules-based system, governance, policy or process. It is important to note that governance can occur not only in recognised management boards but also on more operational levels that also set out priorities, monitoring, control measures and provide input for the context that management implements strategies on. So a decision on whether to follow a specific framework or not is also a governance strategy if there is an overall aim or long-term goal driving this (set of) decisions.

## 2.2 Literature review methodology

The methodology of Kable et al. (2012) was used to conduct the literature review. The purpose of this literature review was to identify if there exists a relationship between architecture and agility. Additionally, it was conducted to identify knowledge gaps, which can be used to formulate a research question. With this purpose a literature search was carried out in Web of Science, according to Table 1. Papers were selected using inclusion and exclusion criteria (Figure 4), the selection process is added for transparency and reproducibility in Appendix A in Table 15.

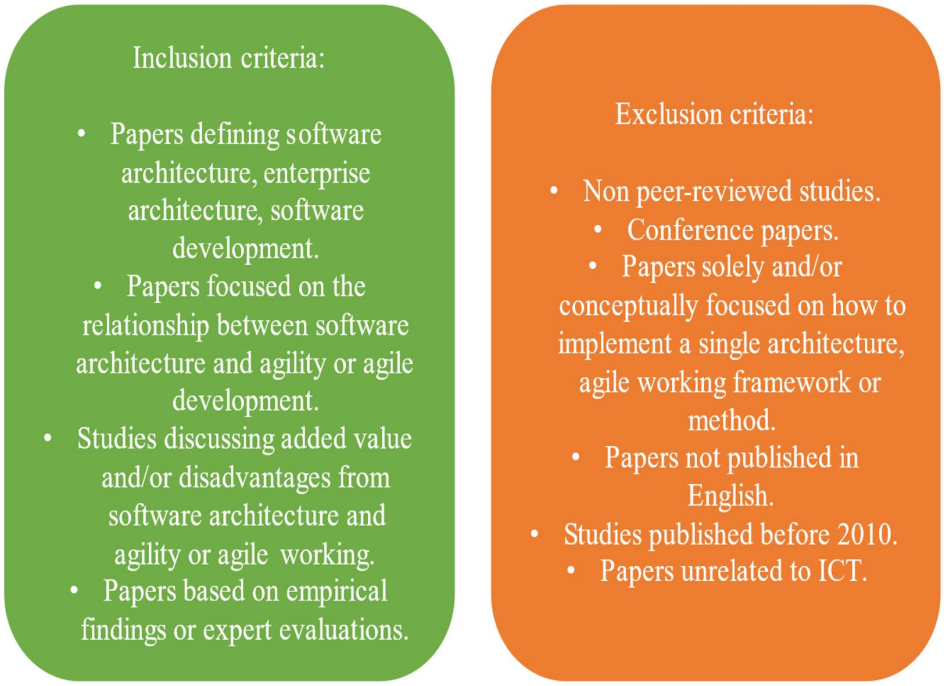


*Figure 4: Inclusion and exclusion criteria*

Due to the very large number of hits for the first two search terms, the papers were filtered on their type and journal type to reflect the inclusion and exclusion criteria: review article and journal article; business, multidisciplinary and computer science related journals. Search by hand was then performed on the papers that were indicated as highly cited by Web of Science. Search term three was fully searched by hand. The papers that were selected by hand were then conducted to quality appraisal. The quality assessment was done by reading the abstract of all remaining papers and scanning the text and citations for keywords to identify whether the paper discussed a relationship between architecture and agility. During this quality assessment the focus narrowed from enterprise architecture to software architecture, as this was identified to be more relevant in combination with agile development and most papers that were found discussed this combination. Replacing, “Enterprise” for “Software” in the most important search term (3) yielded the same results, most likely because of the connection with AND between search query 1 and 2, as search term 2 contains the word software. An overview of the selection process is given in Figure 5.

*Table 1: Search overview*

| Search query | Database and hits | # met inclusion criteria | Article ID in other tables |
|---|---|---|---|
| (TS=((“Architecture” OR “Enterprise architecture” OR “architecting” OR ”*architecture” OR ”Enterprise?architecture”)) AND TI=((“Architecture” OR “Enterprise architecture” OR “architecting” OR ”*architecture” OR ”Enterprise?architecture”))) | Web of Science,<br><br>81,348 | 3,329 | Not applicable, highly cited papers were all architectural designs focused on a specific application. I.e. for a new technology. |
| (TS=((“Agility” OR “agile” OR “agile*” OR “*agility” OR “agile?working” OR “agile?software” OR “agile?development”)) AND TI=((“Agility” OR “agile” OR “agile*” OR “*agility” OR “agile?working” OR “agile?software” OR “agile?development”))) | Web of Science, 13,352 | 937 | 4 papers selected from 9 highly cited papers, 5-8. |
| [search query 1] AND [search query 2] | Web of Science, 32 | 10 | 11, 12, 14-18, 23-25 |

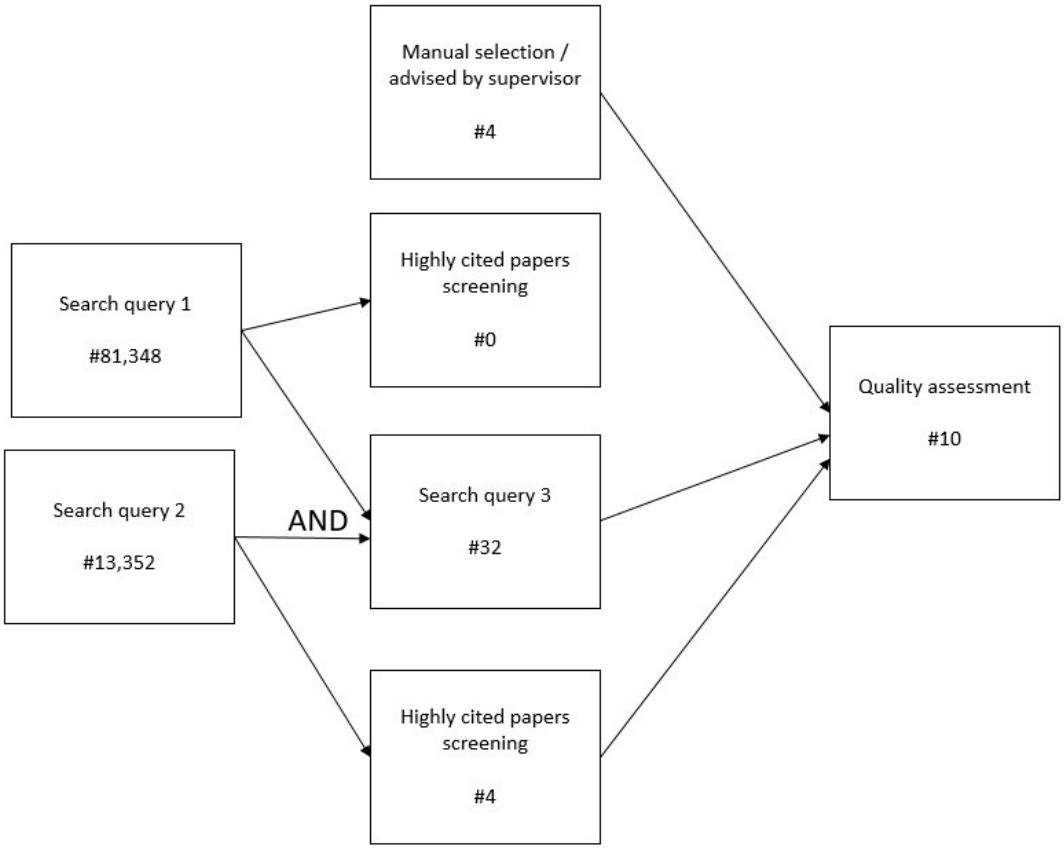


*Figure 5: Literature selection process*

## 2.3 Literature review

Table 3 provides an overview of the papers that were selected. It can be used as a shortcut to identify which information came from which study and as a reference for how the study under review was conducted. Table 3 can be found at the end of this section. The reviewed papers are classified into four categories: review articles, agile practices that can help architects, architecture practices that can help agile developers and improvements of methodologies for the combination. Table 2 shows the distribution of papers under review for each category. Interesting to note is that three papers fall into two categories, these are all papers that introduce or improve a method for the architecture-agility combination. The papers will be discussed based on the classification in Table 2.

*Table 2: Categorisation of themes addressed by papers*

| Authors | Review of architecture-agility combination | Agile practices that can help IT-architects | IT-architecture practices that can help agile developers | New (step in) or improvement of approach for architecture-agility combination |
|---|---|---|---|---|
| Waterman (2018a) | | x | | x |
| (Waterman, 2018b) | | x | | |
| Alsahli et al. (2016) | | | | x |
| Yang et al. (2016) | x | | | |
| Bellomo et al. (2015) | x | | | |
| Woods (2015) | | | x | |
| Poort (2014) | | x | | x |
| Falessi et al. (2010) | | | x | |
| Madison (2010) | | x | | x |
| Gong (2012) | x | | | |
| **Cumulative** | **3/10** | **4/10** | **2/10** | **4/10** |

Figure 6 shows the result of an analysis of the papers considered in the literature review. The green circles on the left side of the figure represent papers included in the review, while blue circles on the right represent papers not included, but similar to the reviewed papers. The analysis was performed using the web-tool researchrabbit.ai. The tool showed that there is a large amount of similar work; 1047 papers. However, most of that predates 2010, which was one of the exclusion criteria. The analysis leads to two conclusions: a) the literature review has a good coverage of the relevant literature of the past 10 years, 16 papers, mostly published around 2010 have not been found using Web of Science or were not selected by hand and; b) research on the combination was peaking from 2000-2010 and receded after this period, as only 10 papers were considered, which covered almost the whole literature that was published in the last 10 years. This begs the question, why has research on the combination receded? As explorative discussions with practitioners did not indicate that problems with the combination are resolved.

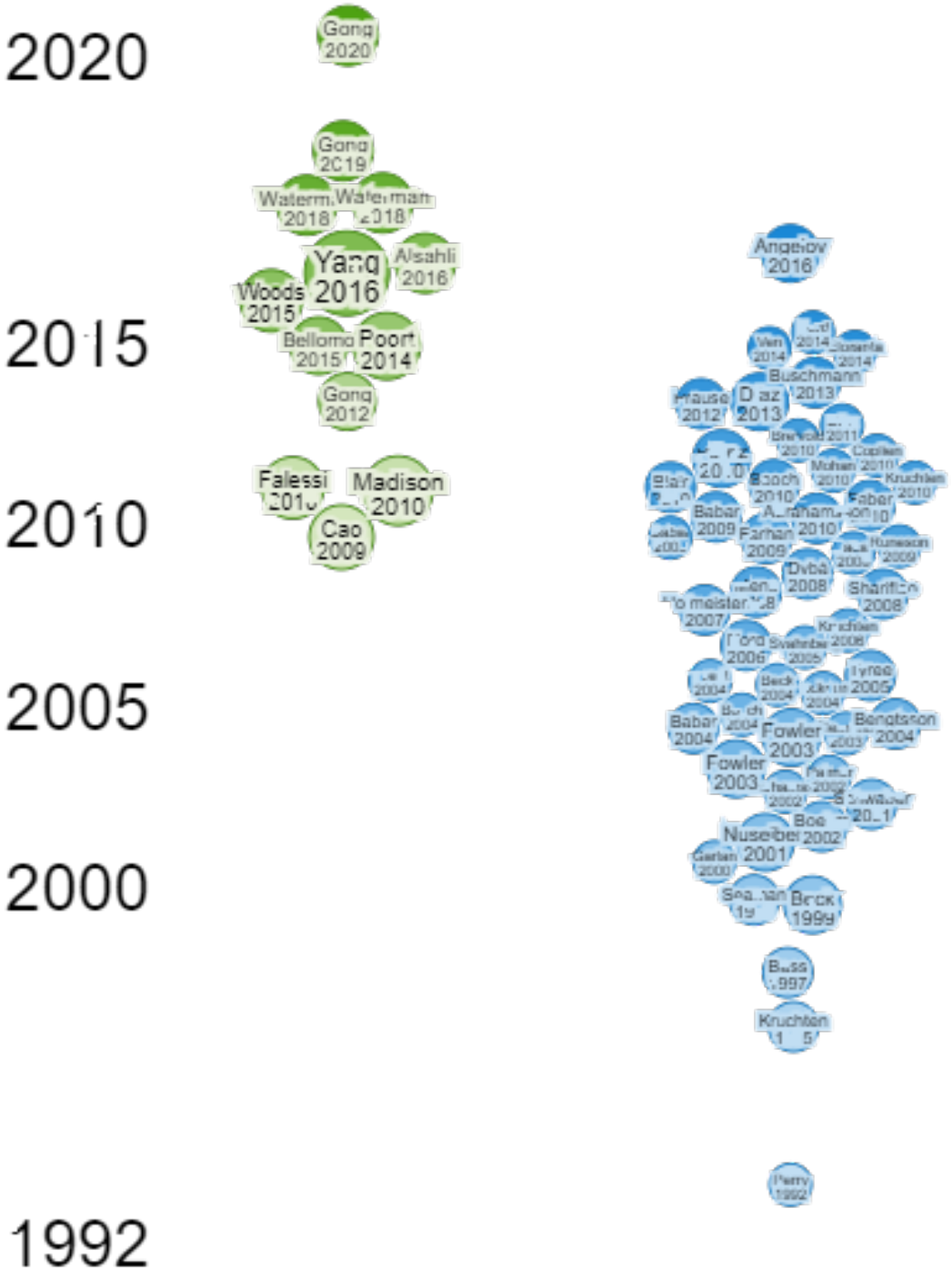


*Figure 6: Timeline of research on the combination*

### 2.3.1 Review articles

A very extensive systematic mapping study for Software Architecture and agile software development has been done by (Yang et al., 2016). It included 54 studies published between 2001 and 2014. The study identified costs, benefits, lessons learned, challenges, approaches, tools and methods. One important finding was that costs and challenges were discussed less in the articles under study. Moreover, all costs were architecture related, which can shed light on why architecture practices are not always seen as added value by agile developers.

It was interesting that there was no recurring architecture approach that was used in the combination. A lack of guidance on how to use practices from the combination has also been identified. Specific tools to support the combination were also absent. The success factors that impact the combination mentioned from high to low frequency were communication, architectural documentation, team, software architect, architectural quality, project size, time, system quality, business and customer. These findings can be interpreted as a lack of governance mechanisms for the architecture-agility combination. Governance could steer on the success factors of the combination.

Moreover, Yang et al. (2016) identified that architectural description, architectural evaluation and architectural analysis are suitable for combination with agile development. Vice versa, the backlog, Incremental Development, Scrum, "Just Enough Work", Sprint, Agile architecting, Continuous Integration and user stories from agile development can be combined with architecture practices. Knowledge of, and action on, these touchpoints can be enhanced by governance.

Unsurprisingly agile architecting is also discussed in Madison (2010), as this paper is also under study in (Yang et al., 2016). The recurring mention of Agile Architecting and work by Falessi et al. (2010) in other studies gives the impression that these ideas caught traction with practitioners and researchers (Poort, 2014; Waterman, 2018a, 2018b; Woods, 2015; Yang et al., 2016).

Another review paper is by Bellomo et al. (2015), their paper analyses which attributes are identified as greatest concern to agile practitioners that used the Architecture Trade-off Analysis. Since a specific method is used in all the studies under review, the results are less generalizable than those in Yang et al. (2016). However, it does show that agile teams struggle with non-functional requirements and that this a good interaction point for the agile-architecture combination, which is in line with the findings of Madison (2010). Another interesting finding is a tension that other articles mentioned as well, that of the short-term functional requirements, which in practice often compete with medium- and long-term quality goals (the non-functional requirements, i.e., scalability, security etc.).

Gong (2012) designed a reference architecture that is focused on agility and flexibility. In order to do so, they defined the concepts of IT-architecture and agility as well as their relations. Moreover, they created a method to measure agility and flexibility, using the surrogates operational performance, time, cost and quality. Their surrogates could be used to identify added value and problems. Their combination of literature review and case-study approach could be taken as an example for this research. They also state that architecture governance is needed to address a long-term perspective of their reference architecture. This statement underlines the importance of governance in the agility-architecture combination.

### 2.3.2 Agile practices that can help IT-architects

Four studies under review help IT-architects to add value in agile development teams and/or environments. Madison (2010) proposes Agile Architecting, a way of working where IT-architects leverage four critical interaction points with agile development teams: Up-front planning, Storyboarding and Backlogs, Sprint participation and Working Software delivery. He combines these interaction points with four architectural skills that he finds most relevant: Decomposition into Sprintable Form, Advocacy with the product owner, IT-architecture backlog and incremental enterprise architecting. Interestingly, most of the advice focuses on involvement of the IT-architect in the agile development team(s) and communication with the teams and product owner. Yang et al. (2016) identified communication, the team and software architect, customer and business as important factors that impact the agility-architecture combination. Thus, it can be concluded that the proposals by Madison (2010) are confirmed by Yang et al. (2016) in this extent.

Waterman (2018a & 2018b) were written like a diptych. The first part discussed a dilemma that software architects face when operating in an agile environment: How much up-front architecture design needs to be done (Waterman, 2018a). He pointed out that the balance point is determined by contextual factors, such as the friendliness of the organisation for agile development as discussed in section 2.1.2. He defines two dimensions of an agile architecture:

1) The (software) architecture has been designed using an agile process.
2) The (software) architecture is modifiable and tolerant of change.

In other words, the software architecture and the decisions that the software architecture is based upon should not be static. The software architecture should be adaptable, able to evolve over time and various iterations. This seems logical in a small business environment or a green field situation, however, it is questionable if this is achievable brown field situations or in a large or regulated organisation with more stringent architectural requirements. Still, Waterman proposes five tactics to design agile architectures:

1) Keep designs simple: reduces the up-front effort, increases modifiability.
2) Prove the software architecture with code iteratively: reduces up-front effort, increases modifiability.
3) Use good design practices: can increase up-front effort, increases modifiability.
4) Delay decision making: reduces up-front design effort, increases tolerance to change.
5) Plan for options: can increase up-front effort, increases tolerance to change.

In his second work Waterman (2018b) addressed how risk affects the amount of architecting that developers must perform up-front. Architectural work up-front is a way to reduce this risk, however it is context dependent how much up-front work is necessary to reduce risk and the underlying uncertainties to manageable levels. High failure probabilities were identified by participants to be often caused by complexity in terms of:

1) Scale, the number of things considered,
2) Diversity, the number of different things,
3) Connectivity, the number of relationships between things.

The research showed that reducing risk negatively impacts a team's ability to design an agile architecture. As the team needs to reduce risk, more up-front design is done and decisions are made early. Therefore, tactics that aim to postpone decisions and keep designs simple are less effective. Thus the balance point of up-front design of architecture is a balance between agility and risk. Teams need to reduce the risk to a level that is satisfactory to the team and stakeholders. That amount will determine the ability to design an agile architecture and in turn the ability to deal with unstable requirements. This is interesting, as governance strategies can be used to identify and control risk (Chulani et al., 2008). Moreover, governance and management are often determinant for the risk tolerance of an organisation. However, Waterman does not discuss which roles and responsibilities are involved at various stages of this agile architecture or which monitoring, and control systems could be used to identify, assess and mitigate these risks.

Poort (2014) identified five ideas to work effectively as an IT-architect in an agile environment. Recurring elements from other studies were the architectural backlog, keep it small and just enough anticipation (Madison, 2010; Waterman, 2018a; Woods, 2015; Yang et al., 2016). New was the idea that decisions are the main deliverable of an architect, not documentation, and that economic impact should determine the focus of the architect. This can be expressed in risk and costs, in line with the work by Waterman (2018b) and Chulani et al, (2008).

### 2.3.3 IT-architecture practices that can help agile developers

Two studies (apart from the reviews) discussed architecture practices that can help agile developers. Falessi et al. (2010) stated that the main problems in combining agile and architecture-centric models are not theoretical but reside in practical matters of adoption. Unsurprisingly this has been found by Yang et al. (2016) as well, as Falessi et al. (2010) was under study in Yang et al. (2016). Moreover, they found a willingness of agile developers to use architectural design patterns for integration of architectural practice into agile methods. This is interesting as it has been found that non-agile developers overstated the contrasts between architectural and agile approaches.

Woods (2015) proposed to link five responsibilities of an architect to the four values of the agile manifesto. These five responsibilities are:

1) Focus on design work,
2) Meet the needs of a wide stakeholder community (beyond users and acquirers),
3) Address system wide concerns (often non-functional),
4) Balance competing concerns to find acceptable solutions to design problems, and
5) Provide the leadership required to ensure that the system's architecture is well understood and supports its successful implementation.

The proposals by Woods (2015) can be recognized in Yang et al. (2016) as well, even though this study was not part of the systematic mapping study. For example, the importance of communication for success is identified by both studies. Moreover, it underlines the importance of non-functional requirements or system qualities in the agile-architecture combination identified in other studies under review (Bellomo et al., 2015; Madison, 2010; Poort, 2014; Waterman, 2018a, 2018b).

### 2.3.4. Improvements to methodologies

The Twin-Peaks model aims to integrate requirements engineering[27] and change management of requirements[28] with software architecture (Cleland-Huang et al., 2013). Alsahli et al. (2016) further developed the Twin Peaks model by combining agile case-based reasoning (CBR) and the Twin-Peaks model. Case based reasoning is a practice from AI which stores cases in order to reuse them later in similar situation. The goal is to create and update a knowledge base repository which can be referred to in new situations. They found that doing so can lead to better synchronisation of requirements and software architecture during global software development. However, with the way the questions are posed, a critical reader might ask how an expert should assign a percentage to things like increased complexity or how much agile practice lacks synchronization between requirements and software architecture during global software development. Making the generalisability of the findings questionable. Nevertheless, the idea of implementing the idea of CBR with the idea of Poort (2014), where software architecture decisions are deliverables of software architects in order to achieve agility, would be interesting. Especially, since Yang et al. (2016) identified experience and personal knowledge as important factors for the application of agile practices in the architecture-agility combination. Similarly, (Waterman, 2018a) identified the team's architectural and technical experience as contextual factors that determine the balance point in the amount of up-front architecture that is needed. These findings establish a theoretical basis for the idea that CBR could be implemented for architectural decisions in order to improve the access to architectural knowledge, which can enhance the team's application of the architecture-agility combination. More importantly, these findings show that a multi-case study approach can contribute to science and practitioners as these cases are an update of the publicly available, scientific knowledge repository.

[27] Agile software development addresses the gaps in interactions between developers and the customer, one of which is requirements engineering.

[28] Speed in adoption of changing requirements and corresponding changes in architecture is an issue in the combination.

*Table 3: Overview of selected studies*

| ID[29] | Author | Purpose | Data sample | Design | Key findings |
|---|---|---|---|---|---|
| **11.** | Waterman (2018a) | Address the "how much architecture up-front" dilemma agile development teams face. | 44 agile practitioners | Propose 5 principles to develop agile architectures. | Five principles to develop agile architectures and to address the "how much architecture upfront" trade-off in agile development teams. |
| **12.** | (Waterman, 2018b) | Examine how risk affects the amount of architecture that developers must perform up-front. | 44 agile practitioners | Identification of an issue. | There is a trade-off between risk reduction and agility within the up-front architecture design trade-off. |
| **14.** | Alsahli et al. (2016) | Introduce an innovative approach to handle requirements and architecture changes simultaneously (Twin Peaks) during global software development. As current tools or approaches are lacking. | Literature review and 20 domain expert interviews | Approaches uses Case-Base Reasoning and agile practices. Grounded theory, statistical analysis for data analysis | Agile case-based reasoning, when merged with the Twin Peaks model, results in better synchronization and change management of requirements and architecture during global software development. |
| **15.** | Yang et al. (2016) | Identifying evidence on various aspects of – and finding gaps in the application of architecture in agile development and the other way around | Literature study on using software architecture in agile development as well as using agile methods in architecture-centric development. Large portion of studies are conducted in an industrial context. | SMS aims to map evidence at a high level for a specific, but broad topic. | There are many architecture-agility combinations possible. Challenges and costs of architecture-agility combination are less present in literature than benefits and lessons learned. Communication is the most important factor that influences the success of the combination. Several challenges are identified that can be used as research gap. |
| **16.** | Bellomo et al. (2015) | Gain insight in the quality attributes of greatest concern to agile practitioners. | Agile practitioners working with the Architecture Trade-off Analysis Method from the past 15 years over 31 projects. | Data analysis of two studies that analysed ATAM scenario data. | Major concerns are: inclusion of modifiability, performance, availability, interoperability and deployability. |
| **17.** | Woods (2015) | Provide some general advise to help architects align their work with agile teams. | Own experience | Link values from agile manifesto to architecture practices that support agile development. | Architectural practices can support agile development. |
| **18.** | Poort (2014) | Offering five pieces of advice to help architects become more effective in an agile environment without having to implement new methods or frameworks. | Own experience | Present ideas instead of fully worked out practices or principles. | Five ideas themselves. |
| **23.** | Falessi et al. (2010) | Explore theoretical compatibilities between agile values and software architecture | 72 IBM (agile) software developers working in Rom, average of 18 years of experience | Quantitative data captured by conducting focus groups and synthesizing comments into a survey to capture quantitative data. | Agile software developers have a the perception that software architecture can contribute in a positive way to their own practices. Either always or in complex situations. |
| **24.** | Madison (2010) | Proposal of an approach that uses agile techniques to drive towards good architecture. | Own experience of working as an architect in agile teams. | Set of interactions and critical skills. | The approach requires an architect who understands agile development, interacts with the team at well-defined points influences them using critical skills easily adapted from architectural experience with other approaches and applies architectural functions that are independent of project methodology. |
| **25.** | Gong (2012) | Design of a conceptual architecture that provides agility and flexibility | Interviews and document analysis, IND (case study subject) employees | Literature review, single case study, design science and prototyping | Example of a case study related to the architecture-agility combination. Measure to expresses agility and flexibility. |

[29] Reference to the selection process in Appendix A.

## 2.4 Knowledge gaps & Research question

This section summarises the main complexities and research gaps found in the literature review. Software architects deliver documentation of the design and quality attributes of the projects (Woods, 2015). Preferably up-front to reduce risk (Waterman, 2018a, 2018b). Therefore, software architecture iterations tend to be longer than those of agile software development teams. Agile software developers tend to give priority to requirements directly functional to the customer and working code delivered in short iterations (Beck et al., 2001b). Therefore, good touch points for the architecture-agility approach are the iteration length and non-functional requirements (Bellomo et al., 2015; Madison, 2010; Poort, 2014; Waterman, 2018a, 2018b). Interesting factors to collect data on are mitigation of uncertainty or risk, communication, team and the software architect's characteristics, perceptions and knowledge.

The literature review shows that how and when IT-architecture and agility can complement each other is perceived as interesting by researchers and practitioners. However, the amount of research done on the subject is limited and not every possible combination has been discussed yet (Yang et al., 2016). Moreover, a lack of tools, frameworks and methodologies has been described, hence various authors have tried to design their own or improve those of others, as shown in Table 2. This table also shows that the majority of advice was focused on how architects or agile development teams can obtain added value through the architecture-agility combination. Interaction points, ideas, principles and approaches have been identified in the literature review. However, these are often theoretical and abstract. It is unclear how these ideas can be correctly used in practice. There was mention of success factors that could be influenced by governance mechanisms, however often not explicitly named in this context (Gong, 2012; Madison, 2010; Poort, 2014; Waterman, 2018a; Woods, 2015; Yang et al., 2016). Nevertheless, the reoccurrence of factors like communication, participation, people, resources and project size impacting the success of the architecture combination asks for a governance approach that addresses these success factors.

Another interesting gap in knowledge are the problems that can occur in the combination, as the focus of most studies under review was on the added value. However, focussing on the benefits only does not do justice to practice (Waterman, 2018a; Yang et al., 2016). Therefore, more research that addresses problems for the combination would add to theory. Another interesting gap is how roles and responsibilities are defined, which interaction models they create or break and whether they lead to added value or problems. The same can be said for monitoring, control and communication. Therefore, the following research question is central to this thesis: *How do governance strategies help to obtain complementary added value from the interaction of IT-architecture and agile software development?* Or more simply put, how do governance strategies relieve the tensions between agile software development and IT-architecture? This thesis will address this research question through a theoretical perspective, as well as an empirical perspective. A framework on how the interactions can work from a theoretical perspective will be compared to how the interactions work in practice. Chapter 3 will discuss how the research question will split up into sub questions and how these sub questions will be addressed.

# 3. Materials and Methods

This section starts with the chosen research approach, sub-questions and approach per sub-question. Then, case definition and case section will be presented. After which data analysis methods are discussed. Benefits as well as limitations of the materials and methods are considered. Procedures and instruments are discussed in detail in Appendix B.

## 3.1. Research approach and data collection

Since the main research question focuses on interactions between approaches employed by people, a more qualitative research approach is suited (Creswell, 2009). Moreover, the data collected on added value is based on human understanding that is ascribed to the architecture-agility interaction this human understanding is, according to Creswell (2009), at the core of qualitative research. In the previous section it was identified that the amount of (recent) research done on the subject is limited and that there is a lack of knowledge on frameworks, tools and methods to structure working with the architecture-agility combination or interactions in scientific literature. Additionally, it was discovered that research specifically on governance strategies that help to obtain complementary added value in the architecture-agility interactions is novel. Moreover, little advice on how to apply these frameworks or tools that were designed for the combination has been identified. Breivold et al. (2010) and Yang et al. (2016) stated that research that has been done on the interaction is often of small size, scattered and of limited generalisability as there are no agreed upon metrics by researchers of the interaction. They argue that further research based on defined metrics on in an empirical setting is necessary to understand the interrelations of architecture and agile development. Thus, combining a theoretical and empirical perspective to determine if, how, where, when and why agile software development and architecture are complementary or not would add to scientific knowledge.

Explanatory and exploratory research is well suited for a situation where little theory is available and where the goal is not to conclude a study but to develop ideas for further study. Additionally, qualitative research allows for rich descriptions, needed to formulate theory (Mintzberg, 1979). A multi-case study method inspired by Eisenhardt's (1989) approach to build theory from case studies was used. Eisenhardt (1989) identified situations where little empirical substantiation is present for theory (i.e. frameworks) as suitable for her approach. Case studies were chosen over interviews as interviews leave the expert to describe the questions only, while case studies allow for a more complete picture and validation by means of triangulation (Yin, n.d.; Yin, 2018). Even though, conducting multiple case studies is more demanding, usage of multiple case studies allows for new insights to be tested and increases validity and generalisability. The method draws from other theory building approaches that use case studies, such as 'grounded theory' from Glaser and Strauss (1967), Yin (1981, 1984) and Miles and Huberman (1984). The used research approach was of an inductive, explanatory nature as the research tried to formulate theory based on empirical findings (Creswell, 2009; Eisenhardt, 1989). As opposed to deductive, experimental research that looks at facts in order to confirm or reject a hypothesis (Bourgeois & Eisenhardt, 1988). The approach is suited to be used in order to examine various facets, the how and why, of a causal argument (Eisenhardt, 1989; Yin, n.d., 2018). The main research question was a 'how' question, aimed at discovering ways of achieving added value through governance from the interaction of two software development roles. Hence, a combination of exploratory and theory building focused multi-case study approach was considered suitable and chosen to address the research question.

An initial theory, based on the literature review, is that governance strategies affect the interaction of IT-architecture and software development roles and that complementary added value or problems can be obtained from these agile-architecture interactions. As the empirical interactions, governance strategies, added value and problems were not yet known and needed to be identified, the research was also of an exploratory nature, defining a typology of interaction models. Then these explored factors can be connected through one or more theories on why and how these interactions models add complementary value or not.

## 3.2. Sub questions and approach per sub question

The main research question is split into four sub-questions:

1. How are IT-architecture and agile software development interactions described in academic and grey literature?
2. How can empirical, public sector IT-architecture and agile software development interactions be classified according to interactions from academic and grey literature?
3. How do context factors impact these empirical agile-architecture interactions?
4. What are reinforcing and balancing problems for the empirical interactions of IT-architecture and agile software development?
5. What complementary added value was experienced in empirical interactions of IT-architecture and agile software development?

The first research questions aims to identify possible interaction models from academic and grey literature. While some articles discussed the interactions of IT-architecture and agile software development as a combination, there could also be more separate IT-architecture and agile roles and approaches involved in software development. For this it is interesting to first look at grey literature, such as frameworks, for theoretical interaction models, as these ideas guide practitioners and are likely to influence the empirical interactions that take place in practice. Based on this an initial theoretical framework, devised of several conceptual interaction models can be created. Then for the second research question, empirical interaction models can be distilled from the case studies. After which, the empirical interaction models can be classified according to the conceptual models, the conceptual interaction models can be validated in this way and improved if necessary. The interaction models are no longer conceptual, but empirical after this step, as they stem from the case study data. The third research questions uses factors found by literature to investigate how context factors affect the case studies. These factors are risk management, communication, knowledge. The fourth research question helps to identify problems in the form of bottlenecks and tensions. Bottlenecks are limitations to a desired outcome. While tensions are differences in perspectives of different roles that complicate the software development process. These differences in perspectives could stem from the responsibilities and tasks a role is given by governance strategies, but could also stem from other phenomena. It would be interesting to know whether these tensions or bottlenecks are reinforcing other problems or providing a balancing effect with another problem, for example as a trade-off with a less desirable problem. The fifth research question aims to identify added value and whether there is a complementary nature in the added value that is found; for example by one benefit reinforcing another, but only when two phenomena are present together. This complementary added value can then be linked to interaction models and governance strategies to answer the main research question.

## 3.3. Data collection and processing

Data was collected by using semi-structured interviews and document analysis. In this study data collection was practiced by setting up separate interviews with individual roles, inquiring the interviewees on their own role, as well as the roles they interact with and then comparing an interview with one role with another. Thus, allowing for triangulation of statements made by participants. This data collection setup allows the researcher to compare multiple perspective on governance strategies, the software development process and their outcomes.

Data has been collected by interviewing twelve participants over seven different organisations. Microsoft Teams has been used as a tool to conduct and record the interviews. Transcription and analysis has been performed by hand. Coding of the interview transcripts as well as organising data in tabular displays were means to process and analyse the data in a systematic way (Yin, n.d.). Google has been used for document analysis and to identify grey literature. Google Scholar and Scopus have been used to find academic literature outside of the literature review.

### 3.3.1. Limitations of data collection methods

Documents analysis and semi-structured interviews were the chosen data collection methods and their benefits and limitations have been considered. Data collection field procedures are discussed in Appendix B.1. Interviews are prone to response bias and bias due to poorly articulated questions (Yin, 2018). Inaccuracies can arise as respondents have trouble recalling events and participants might say what the interviewer wants to hear, this phenomenon is called reflexivity. Thus, the way the questions have been posed and the interviewer's behaviour during the interview could have caused bias.

#### Pilot

To cope with the limitations of interviews as a data collection procedure, a pilot interview was performed. This helped to assess and sharpen the interview questions and interviewers behaviour, for example by asking more explicitly for roles in the software development process. The pilot interview led to the deletion of one question that was irrelevant. This question took away time from other questions as the participant had trouble understanding the question and started to question the researcher what he meant; hence the question was ill-formulated. The next question was better formulated and covered the same subject of the deleted question. Most importantly, the pilot interview (and later interviews) showed that participants have trouble to answer questions on the interaction of agile software development and architecture. Participants responded by giving advantages and disadvantages of the other role, so for example, an agile development team member gave benefits of how their IT-architect did things at first. However, this problem was circumvented by interviewing both IT-architecture roles and agile development team roles that worked on the same project and by asking the participant why the added value could not be attributed to IT-architecture and agile alone. This way the case studies were still able to form a description of the interaction based on the perspectives of both parties in the interaction. Interestingly, interviewees with more experience in their role were able to give advantages and disadvantages specific to the interaction of software architects and agile software developers.

Another finding of the pilot interview was that participants struggle with defining a governance strategy and that examples helped the participant to think of governance strategies that influenced their own projects. To limit the chance that the respondent responded with an answer similar to the example[30], multiple examples are given. However, this bias cannot be fully eliminated. Moreover, respondents[31] doubted whether they gave satisfactory answers when asked to describe how governance strategies affected their work. Even though, their answers did describe governance strategies and how they affected their work. Finally, the pilot showed that 1,5 hour of interview time was sufficient and that the questions that were formulated provided the researcher with useful data for the research questions.

#### Other coping mechanisms

Control questions have been asked, inquiring on benefits as well as disadvantages, how the project was delivered in terms of budget, time and quality. Documents are biased as they are drafted with a specific purpose or audience in mind (Yin, 2018). This is especially the case for governance related (i.e. policy) documents, as these are likely to highlight the envisioned positive effects, while 'downplaying' the (possible) negative consequences. Supporting[32] documentation has been difficult to access due to the confidential nature of the topics that were discussed in the interviews. In analysing documentation, the (unknown) bias of the author and biased selectivity have been considered. Rival explanations have been sought out deliberately. Since most organisations were public, governance related documents were publicly accessible in some cases. However, due to the confidentiality that was promised to the organisations these could not be cited in this thesis. Thus, being of limited use.

[30] A type of bias identified as anchoring and the availability heuristic by Kahneman.
[31] This also happened in interviews after the pilot.
[32] Or conflicting

### 3.3.2. Benefits of data collection methods

On the contrary there were also advantages to consider. Document analysis is repeatable, unobtrusive, specific (i.e. on event) and can cover a long-time span or consider many events or settings. In many cases document analysis did allow the researcher to place responses by interviewees in context or to falsify or validate doubts on claims made by participants. Agile and IT-architecture frameworks and ideas have been analysed through document analysis, which also helped to assess claims made by participants.

The interviews also had benefits, this data collection methodology allowed the researcher to be flexible in data collection, and to explore new insights and rival explanations needed for theory building. The first interviews were more explorative towards software development practises and roles in general. In later interviews more attention was given towards insights and theories based on earlier interviews. This prolonged these interviews, with the longest interview almost reaching 2 hours. Moreover, the interviews were held with an individual participant, this meant that the participant could be critical on their own and the other role in the interaction. Interviewing both roles individually allowed the researcher to pose statements by one participant of the organisation to the other participant, which represented another role in software development. This helped to establish perspectives and seek conflicting explanations. Finally, the interviewees allowed to 'deep-dive' into certain aspects with different roles in software development. For example, interviews with enterprise architects tended to be more focused on the organisational impacts and governance aspects, while interviews with developers or solution architects focused more on operational aspects of software development.

## 3.4. Data analysis methods

Four general data analysis strategies are identified by Yin (2018): 1) relying on theoretical propositions; 2) working data from the 'ground up'; 3) developing a case description; and 4) examining plausible rival explanations. This research is best identified with the second strategy: working data from the 'ground up' as this strategy is most in line with Eisenhardt's (1989) methodology for theory-building through case studies. While a theory building case study may seem linked to the first: relying on theoretical propositions it is important to note that these propositions are currently underdeveloped,[33] and therefore, cannot be tested, which is at the core of this general strategy. However, for future research, if a theory has emerged from the data, this strategy becomes more relevant to test the robustness and validity of the emergent theory. Similarly, developing description of software development interactions, governance strategies and added value that impacted them helps, but is not the main goal. These descriptions are extracted to facilitate the theory building process. The same can be said of examining rival explanations, which is another important method in theory-building. Thus, the general analytic strategy of working data from the ground is most in line with the chosen approach by Eisenhardt (1989).

Moreover, Yin (2018) identifies five analytical techniques: 1) pattern matching, which can be done for a) processes and outcomes; and b) rival explanations; 2) explanation building; 3) time series analysis; 4) logic models on a) an individual level; b) organisational level; and c) program-level; 5) cross-case synthesis. Logic models were not considered fit for this research, as they require a specific theory, intervention or change to analyse. This data analysis method is more suited requires a specific theory to be tested and described in a logic model. The fifth data analysis method, cross-case analysis, will be discussed in a later, separate, section. Pattern matching and explanation building seemed to be most relevant in respect to Eisenhardt's (1989) theory building methodology and thus were used in the research. How these analysis methods were used is explained in more detail below. Note that pattern matching is discussed as it is similar to explanation building and some elements of pattern matching were also used in explanation building, but that the main data analysis method was explanation building. The within and across case analyses are discussed separately.

[33] There are some ideas, identified in the literature review. However, these are high-level and practical guidelines on how this can be implemented are lacking or limited. Moreover, the link with governance strategies is often implicit, based on the researchers own deductions.

### 3.4.1. Analysis of within-case data

For the case-study data, it is up to the researcher to define the codes or procedures that are used to logically piece together the coded evidence into broader themes (Yin, n.d.). One way to do so was by systematically organising words and narratives into tables. Then the researcher performed different types of analysis like explanation building, pattern matching. In the multi-case study, these analyses were conducted within each single case first, after that the replication logic technique has been applied to generalise findings analytically[34] (Yin, n.d).

A first step in data analysis was to play with the data. This was done by juxtaposing the data from two different interviewees, putting information into different tables reflecting different themes and by writing memos on observations in the data to oneself (i.e. through comments in word, notepad and a separate analysis document for each interview). Coding was be used with colours, allowing to easily identify related concepts[35] in the case study database. This was done by hand.

#### Pattern matching

Pattern matching was be used to compare a predicted pattern with an empirical one (Yin, n.d.). Thus, this was a useful tool to compare the theoretical findings of the literature review with the findings from the empirical data. Pattern matching is a powerful tool in explanatory case studies, as the patterns might relate to 'how's' and 'why's' within the cases studies (Yin, 2018). It is important to avoid address very subtle patterns, as these could easily be challenged based on interpretation. These patterns can also be identified within single cases first and then across cases to increase validity of the process-outcome relationship. Similar to Yin's (2018) description of this method, not all patterns were be predicted through literature, but have resulted from earlier case studies.

Pattern matching of rival explanations is an important tool to satisfy validity of theory. However, it required the researcher first to identify conflicting propositions and then to set up different case studies in such a way that they predict conflicting or matching outcomes in order to test and match these outcomes with the conflicting explanations. This was certainly important in theory-building, however, since it required defined contrasting propositions, this method for pattern matching was difficult to establish in a field where theory is lacking. On the other hand pattern matching of rival explanations was interesting for conflicting explanations that emerged from individual case studies.[36] Then a case study was set up, especially to test a rival hypothesis.

#### Explanation building

Explanation building is suited for explanatory studies and especially of great interest for case studies whose explanations reflect some theoretically significant propositions (Yin, 2018). Explanation building was used to analyse the findings of the main research question, as the objective of this question is to build and falsify theory about the influence of governance strategies (i.e. policy) on complementary added value derived from combining IT-architecture and agile software development.

The process of explanation (theory)-building can be operationalised as follows: 1) making an initial but provisional statement or explanatory proposition (theory/explanation); 2) comparing the data from your case study against such statement or proposition; 3) revising the earlier statement or proposition; 4) comparing other details of the case against the revision; 5) if doing a multiple-case study, comparing the revision from the first case with data from other cases, leading to further revisions; and 6) repeating this process with other cases as many times as needed (Eisenhardt, 1989; Yin, 2018). This procedure has been followed. Note that this is an iterative process and that comparison with relevant

[34] Not statistically

[35] Agile, architecture, interaction, governance, project specific context, advantages or added value and problems or disadvantages.

[36] A nice example is the proposition that agile is not suited for a government context. For which the private sector case study was set up.

literature[37] played a constant role in this process.

The key difference with pattern matching is that the final explanation may alter from the starting theory that is developed at the beginning of the study. Yin (2018, p. 231) stated that: *"The clearest result would be if your case study data do not support these rival explanations."* Consequently, entertaining rival or plausible explanations or theory was very important in this case study analysis method as well.

### 3.4.2. Searching for cross-case patterns

An important method for analysing data in this research was comparison of data (explanations) across cases, either rival, as well as similar explanations. In the chosen approach the case study descriptions and data were used to build (and test) theories[38] in an emergent way. Yin (2018) stated that cross-case synthesis allows the researcher to elevate cross-case patterns to a higher conceptual plane, the 'how' and 'why' mechanisms behind phenomena, rather than downward reduction to individual variables. Eisenhardt (1989) seemed to imply the same with the underlying dynamics of the relationships to be studied with his method and stated that case studies can make these observable, which is essential to understand why the relationship exists.

Cross-case analysis helped to cope with humans limited ability to process data. Humans (unknowingly) employ several biases which can lead the investigator to premature or even false conclusions if they base their theory building research on human testimonials (Eisenhardt, 1989). Therefore, good case-cross comparison can counteract these tendencies by looking at the data in divergent ways. Yin (2018) adds to this perspective by stating that in cross-case synthesis it is very important to discuss potentially contaminating differences between individual cases in the multi-case study (i.e. cultural or institutional settings). Thus, literature seems to find consensus on this point and additional questions in the case study protocol were added to inform on the institutional and cultural context of the interviewees and the organisations where the development process took place.

#### Replication logic

Replication logic is an analytical generalisation that requires to first construct a conceptual claim, which can then be applied in another case study[39] where similar concepts are relevant. Replication theory can be used to increase confidence in the validity of relationships with cases that conform emerging theory (Eisenhardt, 1989). Likewise, it was used to refine and extend emergent theory if cases disconfirm relationships. Thus, it helped to understand why an emergent relationship did (not) hold. The essence of replication logic or cross-case synthesis is that cross-case patterns rely on argumentative interpretation not numeric tallies (Yin, 2018).

### 3.4.3. Shaping Hypotheses

Hypotheses were shaped iteratively by comparing the emergent frame with the evidence from each case in order to determine how well it fitted with case data. By constantly comparing theory and data, the result of this process should be a theory that closely fits the data (Eisenhardt, 1989). This is important as this way one takes advantage of new insights based on the data and yields an empirically valid theory. This can be done by: 1) Sharpening constructs; and 2) verifying that the emergent relationships between constructs fit with the evidence in each case.

In order to sharpen constructs two parts of the same process were essential: a) refining the definition of the construct and; b) building evidence which measured the construct in each case. This happened through constant comparison of data and constructs from diverse sources that converged on a single, well-defined construct. In essence, the researcher was establishing construct validity through multiple sources of evidence, in this case literature, documents and interviews. The difference with a traditional experiment or hypothesis testing case study was that the constructs are not specified a priori and thus definition and measurement occurred in an emergent way during the analysis process. Another difference was due to the nature of qualitative evidence, which made it difficult to collapse multiple

[37] In this case this was grey literature on interaction models, frameworks and best practices.
[38] Or propositions made by participants and the researcher.
[39] Or interview, in some cases.

indicators into a single construct, as not all cases had all indicators or indicators differed. A substitute to factor analysis was therefore evidence that was summarised in tables or text.

The second step in hypothesis shaping was verification of the emergent relationships between constructs fit with individual case data. Sometimes relationships could be confirmed by the case evidence, while other times it was revised, disconfirmed or dropped out for insufficient evidence. This happened similarly to traditional hypothesis testing research. Each case served as an experiment in that sense. The difference was that each hypothesis was examined for each case, not for aggregate cases. Consequently, the underlying logic was replication, the logic of treating a series of cases as a series of experiments with each case serving to conform or disconfirm the hypothesis (Yin, 2018).

### Enfolding literature

Literature has a very prominent place in theory building case studies. It is important to consider a broad range of literature. Since academic literature has already been covered in the literature review, new literature analysis was performed on grey literature, such as frameworks, discussions by practitioners on the interaction, best practices etc. Comparing emergent concepts, theory or hypotheses with literature was an essential part of theory building because: 1) if literature that conflicted with emergent findings was found, then confidence in the findings is reduced. Readers may assume the results are incorrect (challenge to internal validity), or are related to specific circumstances (challenge to generalisability); and 2) conflicting literature presented an opportunity. Putting conflicting literature side by side the emergent theory forced the researcher to think in to a more creative, frame breaking mode of thinking, otherwise difficult to achieve. Insights in both conflicting literature and emergent theory were be deeper, while also sharpening of the limits of generalisability of the focal research.[40] Literature that discussed similar findings was also important as this helps to tie together underlying similarities in phenomena normally not associated with each other. This resulted in theory with stronger internal validity, wider generalisability and higher conceptual level. Which was important, as this theory building case study research rested on a limited number of cases.

## 3.5. Definition of a case study, selection criteria

This section will explain how a case study is defined in this thesis and which selection criteria are used.

### 3.5.1. Case study definition

Each case is the interaction of IT-architecture and agile software development in a specific software development process. Thus, not the software development project itself. Information about the project is therefore contextual. Discussing the interaction in only one development process at a time helped to structure the interview and data analysis processes, as this eliminated cross-project contextualities within each transcript. On the other hand, in some cases interviewees could refer to more processes within their current or past organisations, which was helpful to draw comparisons. For example, one domain architect explained how domain architecture was practiced across various organisations he had worked in, which helped to illustrate the interaction model with agile software development in his current organisation. This did not only help with this comparison, it also helped to identify empirical alternatives.

Most importantly, this case definition posed the requirement that at least one IT-architecture role and one agile development role had to be interviewed to get a complete picture of the interaction. This allowed the researcher to compare perspectives on both working practices, added value, problems and governance strategies. Which were of interest as these perspectives have been identified as important factors that influenced the success of the combination in the literature review. In conclusion, the case study concerned a relationship of two roles in software development and its governance and consequences rather than the specifics of the software development project itself, these were context factors that influence the interaction.

[40] This happened for example with the claim that agile cannot work with fixed deadlines.

### 3.5.2. Case study selection criteria

Due to contextualities, the selection of the cases was very important as cases were bounded by time and activity (Creswell, 2009). Selection was not random but informed. Case study selection focused on contrasting cases: at least one success case and an unsuccessful case. The selection criteria could be considered a mix of extreme and critical cases in the typology of Flyvbjerg (2006). A case was selected that was thought to be most likely architecture dominant (due to complexities, i.e. scale or compliance) and a case that was most likely agile dominant (due to high uncertainty and desire to deliver fast or innovate). From these cases I distilled why agility and architecture did (not) complement each other, which complementary added value or problems have been experienced and which governance strategies helped to obtain this added value. Then these cases were compared to more nuanced cases or cases deemed relevant due to new insights.

Especially for a multi-case study where cases were subjected to a cross-case analysis it was important to be mindful of contextual factors that affected the cross-case analysis such as the point in time the case took place, methods, tools, methodologies and approaches used by the actors in the cases (Yin, n.d.). This limitation deserved special attention in selection, as the literature review in this proposal showed that the variety of approaches, tools and methodologies is large. However, keeping contextual factors constant was less important and feasible in multi-case study research than in experimental research. As theoretical insights often stemmed from contradictions, critical or extreme cases. Similarly, both the case study method and theory building from case study method allowed the researcher to alter their data collection and analysis procedures based on new insights, albeit in a scientific rigorous way (Eisnhardt, 1989; Yin, 2018). However, some factors were held constant to structure the data collection and analysis process. The selection criteria from Table 4 were a guiding the case study selection in order increase the generalisability and relevance of the findings. The selection criteria from Table 4 will now be discussed.

#### Time

To ensure the relevance of the results in a fast-changing field of research, the focus was on cases that took place in the past five years. This choice was made as software development practices and approaches are prone to continuous improvement. For example, Extreme Programming (XP) was extensively used in the past, but has been replaced by Scrum (Yang et al., 2016). Software development is a fast phased and innovative topic and practice.

#### Access to expertise and Organisation type

Public organisations were selected, as they are likely more willing to disclose their experiences than private firms trying to gain a competitive advantage through a new software product, which increased accessibility, availability and put less stress on the time schedule of the research. Moreover, many Dutch government organisations are currently transitioning from waterfall software development to agile software development, thus presenting interesting mixed-form interactions patterns.

Organisations needed to employ an agile development team and at least one software[41] or enterprise architect in order to research the interaction between both software development approaches. This was often the case in public organisations as their projects are usually of a large scale. Moreover, the Dutch government is relatively automated or supported by software applications in their public service provision. A limitation of this choice was that organisations have trouble practicing agile development and IT-architecture as standalone practices, thus possibly reducing the validity of explanations on the combination if unrecognised (Gong & Janssen, 2019, 2020). This was also true for public organisations, for example public organisations were sometimes struggling with the adoption of agile working practices. However, if recognised these cases served as extreme or critical cases. Consequently, it was important to identify how agile practises were adopted in a case study and participants were inquired on their agile and IT-architecture practices. This helped to investigate whether

[41] Or solution.

organisations were balanced or more agile or IT-architecture dominant. Moreover, the dominance of either one might have affected the adoption of the other and affected the interaction model.

*Table 4: Overview of selection criteria*

| Criteria | Rationale | Limitation |
|---|---|---|
| Case took place between 2016 and 2021 | A timeframe that is too large may reduce generalisability as materials and methods are likely to differ. Similarly, a timeframe that is too small limits the number of cases that can be considered. | Findings outside scope are not considered. |
| Project should involve an agile development team and at least one IT-architecture role. | The interaction of both practices can only be investigated if both are present in the development process. | Employing both practices requires vast resources. Adopting each individual practice can be problematic in itself (Gong & Janssen, 2019, 2020) |
| Public organisation as product owner | Availability, more willingness to disclose information than private sector product owners and interesting cases. This definition of the criteria allows to interview experts that were contracted to work on a project with a public product owner. | Public organisations tend to be bureaucratic hence the environment can be considered as non-agile friendly. Public organisations tend to struggle with the adoption of agile working practices. |
| No particular geographical scope | Software is often created by a geographically distributed team. | There might be differences in how the combination or separate approaches are employed across geographical regions. Geographically distributed software development is more complex than more centralised development, hence they might be difficult to compare across cases. |
| Development process up to a specific release or development process up to transfer to operational phase | Finished projects are easier to reflect upon. However, software projects are never really finished. Another good point in time is the transition from the development and operation phase. | Agile projects are never finished. Not possible to keep releases constant across cases. |
| Specific agile framework | Keeping the framework consistent increases validity. SAFe is often used within governmental organisations. SAFe has a section dedicated to architecture. | Keeping the framework constant reduces generalisability for organisations that employ other frameworks. Organisations might be working agile in 'name-only' or might be working with mini-waterfalls within sprints. |
| Access to expertise | Comparing the viewpoints of the roles within cases first and then across cases improves generalisability over the viewpoints across cases only as in the latter case there are more (un)known contextual factors that might impact the findings. | Might not be available. |

## Geographical scope

There was no geographical scope considered as software is often created by geographically distributed team (Smite et al., 2020), however focus was on the Dutch public sector. Moreover, development practices might differ over geographical regions which could reduce generalisability. Next to that, geographically distributed software development imposes additional requirements on the development process (i.e. improved communication standards or stable interfaces) as opposed to more centralised software development (Yang et al., 2016). Thus, comparing cases where the geographical scope differed a lot was difficult (i.e. central versus internationally outsourced development).

### Project status

To be able to identify if governance strategies led to complementary added value, it was also important to look at projects that have been under development for a while, or more preferably were already in operational phase. This allowed to participants to reflect and evaluate in hindsight which added value was achieved, as the fruits, added value or problems, often become observable after the project has been finished. Moreover, focusing on development processes that were finished or far underway allowed the researcher to assess whether a certain governance strategy had affected the problems or added value that was observed. On the other hand software is never really finished.

### Agile framework

Keeping frameworks constant increases validity of the findings. However, looking at only one framework reduced generalisability to organisations that employ other frameworks. Therefore, cases with other frameworks were also considered. The SAFe framework is widely used in large software development. SAFe is used by Dutch public organisations to scale agile working and was therefore considered to be a selection criterium at the start. An additional benefit of SAFe is that it includes software and enterprise architecture within its framework. It is important to note that each organisation implemented a framework such as SAFe in a distinct way. However, these variations were seen as welcome since they helped the theory building process.

Since IT-architecture frameworks are not as intensely practiced as agile frameworks, no specific IT-architecture framework was held constant across cases. However, TOGAF and NORA were used as references to identify and discuss architectural working practices and tools that had impact on the combination, in a similar way to the literature review. While these frameworks are focused on enterprise architecture, architecture on a software level is also included as discussed in chapter 2.

## 3.6. Strengths and weaknesses of research approach, data collection and analysis

This section will discuss the strengths as well as the weaknesses of multi-case study research.

### 3.6.1. Strengths

Eisenhardt's (1989) method has a high likelihood of generating novel theory; creative insights often come from putting together contradictory or paradoxical evidence. This can be done across cases, types of data, investigators, and between cases and literature. This research aimed to exploit this strength by using cross-case synthesis and a literature review. Constant conflicting of realities lead to less researcher bias in theory built than incremental studies or armchair, axiomatic deduction. However, Eisenhardt´s (1989) paper was written in a very positivistic way. Which is made clear as a high likelihood of generating novel theory is advertised, while on the other hand overly complex, as well as narrow theory and replication of already existing theory are mentioned as limitations. Thus, the research did not purely focus on explanatory results, but collected data in such a way that exploratory results would be present as a foundation on which theory might be built in order to ensure findings that would contribute to scientific literature. Moreover, case specific knowledge on governance strategies that led to added value or problems can also be useful, as an expert (practitioner) level knowledge is obtained not from general rules, but from many individual cases and nuances (Flyvbjerg, 2006). Thus, even one well-worked out case can be an interesting reference point for similar cases for researchers and practitioners.

For future research, emergent theory is likely to be testable with constructs that can be proven false by future research. Thus, advancing the state of knowledge in the field of software development. Measurable constructs are likely because they have already been measured during the theory-building process. Thus, the resulting hypotheses are likely to be verifiable for the same reason, as verification is constant in the theory building process, while theory generated apart from direct evidence may have testability problems. Finally, resulting theory is likely to be empirically valid, because the theory-building process is tied with evidence so that the resulting theory will be very likely consistent with empirical observation. Comparing data across cases allowed to distil good and bad governance practices which could formulate a basis for governance mechanisms, while keeping the flexibility to discuss the influence of contextual factors and unexpected findings (Eisenhardt, 1989). This and the flexibility to explore new insights that arose during data collection were great benefits of the multi-case study. The

research approach allowed to collect data on actual human behavior and events as well as to capture the distinctive perspective of the participants in the case study. Thus, the chosen research approach allowed to investigate both realist[42] and relativist[43] perspectives. This was interesting as the literature review hints that both perspectives can influence the complementary added value that can be achieved.

### 3.6.2. Weaknesses

Overly complex theory due to the intensive use of empirical evidence might emerge. Good theory has parsimony (Pfeffer, 1982). Since data was large in volume and rich, there was the temptation to build a theory that captures everything. However, such a theory lacks simplicity and overall perspective. It is easy to lose a sense of proportion when a researcher is confronted with vivid, voluminous data. Since qualitative statistical means are often not used, it can be difficult to identify the most important relationships and which relationships are tied to very specific contextual factors. Cross-case analysis can help here, as reoccurrence of patterns that have strong argumentative basis can be an indicator of their importance.

Narrow and idiosyncratic[44] theory can also emerge. Case study theory building is a bottom-up approach, the specifics of the data produce the generalisations of theory. Thus, generalisability might be lacking. Again, the multi-case study approach can help to address this issue. As idiosyncratic theory is unlikely to hold across cases with different contextual factors (i.e. communication and knowledge). However, generalisability might be limited to the public domain, as the private domain has only been limitedly investigated.

Weaknesses can also be found in data collection and have been discussed. A last counterargument is that, during the interview process, both the researcher and interviewee are engaged in a learning process, trying to identify relationships and important factors that affected the object under study in order to get an advanced understanding of the world (Flyvbjerg, 2006). Thus, the understanding of the researcher evolved over the case study process based on new insights and preconceptions were falsified if wrong.

[42] The way power (roles and responsibilities) influences decision making.
[43] If I think something is wrong, then it is wrong.
[44] contextual, fitting one or a small number of instances.

# 4. Agile-architecture interactions according to grey and academic literature: a basic typology

This chapter will start by discussing the clashes and differences of agile software development and IT-architecture that have been found in literature. In this discussion, perspectives from grey literature are also included. Then a first conceptual framework will be introduced. This framework will help to analyse and classify interaction models found in the public sector. Then the perspectives of the TOGAF, NORA and SAFe framework will be discussed.

## 4.1. How are IT-architecture and agile software development interactions described in academic and grey literature?

Table 5 shows aspects of architecture and agile software development that can be interpreted as clashing. The preference for up-front design and documentation of architecture to reduce risk clashes with principle 11 of the agile manifesto, which states that *"the best architectures, requirements and designs emerge from self-organising teams."* (Beck et al., 2001b, p.1). The second clash is the focus on requirements type, functional for agile developers and non-functional for architects. Iteration length is the third clash, as architectural iterations tend to be long as they deliver extensive designs and documentation. Which creates a large amount of rework if requirements change. While agile developers celebrate short iterations to capture feedback early and elicit new requirements early in the process, before too much work has been invested. The final clash is the focal communication method, as agile developers consider face-to-face conversations the most efficient method of communication (Beck et al., 2001b). While IT-architects tend to focus on documentation, as face-to-face meetings are not always the best communication vehicle, as people leave the team for example.

*Table 5: Parts of architecture and agile software development that can be interpreted as clashing*

| **IT-Architecture** | **Associated problem** | **Agile software development** | **Associated problem** |
|---|---|---|---|
| Up-front, large scale IT-architecture design, planning and documentation by architect in order to reduce risk | Resources invested in irrelevant designs and documentation, can't address all risks up-front | Principle #11 of agile manifesto: *"The best architectures, requirements and designs emerge from self-organising teams."(Beck et al., 2001b, p.1)* | Problems that should have been addressed from the start arise later in the development process |
| Focus on system and organisational wide concerns such as quality attributes or non-functional requirements | Non-functional requirements should support the functional requirements, not limit them | Customer satisfaction tends to focus resources on functional requirements | Functional requirements might not work satisfactory for the product owner as supporting non-functional requirements are underdeveloped / lacking |
| Long, plan driven iterations (phases) that address system as a whole (waterfall) | Can cause large amounts of architectural rework if requirements change | Short iterations focused on individual features driven by stories in order to capture feedback early and to welcome changing requirements (short-cyclical) | Products of short iterations might underwhelm customer / product owner expectations and decrease stakeholder confidence |
| Focus on documentation to communicate design and rationale to accommodate independence if the one that build an aspect is not available | Documentation might not be findable or might not be read, especially if there is a lot of documentation, ivory tower or command and control decision making | Principle #6: *"Most efficient method of conveying information to and within a development team is face-to-face conversation." (Beck et al., 2001b, p.1)* | Face-to-face conversations are not always possible due to schedules, geographical locations, sickness, employee turnover etc. |

However, the very aspects that can lead to an interpretation of architecture and agile development as counterparts can also be the theoretical foundation as of why IT-architecture and agile software development have potential to be complementary. As the aspects identified in Table 5 can also be interpreted as how the focus of one approach can address an associated problem of the other, as I will illustrate below at the hand of Table 6, which shows that the similarities of agile and IT-architecture are not present in the approach, but in the deliverables. This conclusion makes sense as both IT-architects and agile software developers want to create working software that adds value to the organisation. However, agile developers tend to focus more on the business value, while IT-architects tend to focus more on the sustainability of the future IT and business landscape, which includes and is affected by working software. The amount of similarities in deliverables is greater than the amount of problems associated with clashing of IT-architecture and agility, is the first hint of complementarity.

*Table 6: Similarities of architecture and agile in deliverables*

| **IT-architecture** | **Agile** | **Rationale** |
|---|---|---|
| Needs IT-architecture work | Needs IT-architecture work | All system need non-functional requirements to support functional requirements |
| Needs to deliver customer / business value | Needs to deliver customer / business value | All system need functional requirement to deliver business or customer value |
| Needs to decide on solutions: i.e. languages, components and principles | Needs to decide on solutions: i.e. languages, components and principles | Choices need to be made either according to a plan or emergently in order to deliver a software product |
| Needs to cope with known and unknown requirements | Needs to cope with known and unknown requirements | All software development processes will have to cope with requirements that are known beforehand and with new and changing requirements |
| Needs to address integration in complex systems or environments | Needs to address integration in complex systems or environments | Complexity can arise from scale, compliance, dependencies etc. and needs to be addressed to deliver a functioning service |
| Needs personal communication as well as documentation | Needs personal communication as well as documentation | Personal communication is very useful to convey ideas, however documentation is important as people fall sick, leave the organisation or change positions. |

While emergent architecture is evidently needed, as unknown unknowns arise during the development process, fully emergent architectures in line with principle 11 of the agile manifesto are not feasible in large organisations or organisations that have high quality standards (i.e. security) due to compliance, integration in the current environment or high system criticality (i.e. hospital systems). Quality attributes or non-functional requirements are needed to ensure alignment with current business, information and application systems and technology, as well as high quality and criticality standards. Introduction of a new software application cannot be allowed to break down other systems and requires up-front planning by IT-architects. On the other hand, the product owner wants a system that meets their functional requirements and wants to see results during the development process. Thus, up-front architecture design should be balanced with self-organising architecture design in order to increase management of known and unknown risks. These uncertainties should be addressed in a long-term[45] economic sense. This balance should be determined up-front with all stakeholders (including IT-architects, product owner and agile team(s)), but open to changes during the development process in order to increase software quality for the organisation.

[45] I.e. Total cost ownership instead of cheapest purchase value.

Long IT-architecture iterations can cause large amounts of architectural rework if requirements change, since IT-architecture designs or documentation tend to be extensive. Therefore, IT-architects might have a natural inclination to hold off changes from the envisioned architecture landscape. While agile developers embrace changing requirements and use short iterations to identify those early in the process. Thus, both iteration lengths might seem at odds, however IT-architects could learn from agile developers in this perspective. They could deliver decisions or outlines of decisions instead of fully finished designs and test them early with the development team, product owner and customer to identify areas of improvement and avoid large amounts of architectural rework later. This can reduce development time, costs and possibly increases software quality as these resources could be spent elsewhere in the development process. Similarly, agile developers need a long-term perspective to work towards. They need to know what the vision for the future landscape is to integrate their solutions properly. In addition, standardisation of functionalities provided by architecture could help them to save time during the development process and help teams to avoid having to reinvent the wheel as architecture tends to look 'over the wall' across various teams.

While documentation can help to convey knowledge if somebody is not available, it cannot be interchanged for face-to-face communication (Beck et al., 2001b). On the other hand, agile developers can avoid problems if a team-member becomes unavailable (i.e., falls sick, leaves the team) by investing time in documentation. Yet again, both practices need to be practiced in a balanced way that is context dependent in order to increase the efficiency and effectiveness of communication.

## 4.2. What do frameworks prescribe?

SAFe is an agile framework that aims to scale agile working practices (Scaled Agile, Inc., 2021). Since it has been found in 2011, some projects in this research did not have the ability to consider the framework, as it did not exist yet or was unheard of at the time. As scaled agile is associated with larger projects, scaling, integration and compliance issues need to be addressed. Consequently, the SAFe framework recognises various IT-architecture roles: that of the enterprise architect, the solution architect and the system architect (the software architect). The framework also recognise the importance of alignment with TOGAF layers, as the IT-architecture roles cut across these layers, which are identified as common domains. In the SAFe framework, solution architects:

1) Design for customer and stakeholders, through understanding the customer and the solution context.
2) Assure feasibility and sustainability, by evaluating emerging technology, partnering with suppliers and creating a continuous delivery pipeline.[46]
3) Design and evolve the technical solution, by using models to describe the system, collaboratively specify the system, decomposing the solution, managing interfaces between components, defining of the solution context, ensuring implementation flexibility, performing technology trade-offs, managing risks and participating in team organisation.
4) Manage non-functional requirements and compliance.
5) Define and prioritise enablers, enablers are supporting elements that are needed for the delivery of functional requirements or evolving the system, such as refactoring or addressing technical debt.
6) Enable continuous delivery.
7) Maintain the architectural runway by creating an architectural vision and roadmap and managing the architectural runway.[47]
8) Manage suppliers by tracking technology across the supplier landscape, selection and evaluation of suppliers and aligning the technical solution across the supply chain.

[46] The continuous delivery pipeline is a based on the DevOps CI/CD practice.
[47] Think of an airplane, it needs a runway to land, and the size of the runway depends on the specifications of the airplane. The airplane is a metaphor for the software solution and the runway for the supporting architecture.

Interestingly, SAFe propagates a balanced exchange model approach to IT-architecture, which is called 'Agile Architecture or the architectural runway.' The framework propagates that a balanced needs to be determined at the start of the project with a percentage of intentional architecture (up-front architecture) and emergent architecture (principle #11 of agile manifesto). This balance needs to be updated throughout the project, depending on the project's needs. The balance point is thus project, time and context dependent. Even within this context the balance point is dependent on the needs of the project at that time and thus changes over time. The framework has no publicly accessible explanations or case study examples that show different reference balance points and implementations.

The rationale for the SAFe exchange model is that an emergent architecture cannot handle the complexity of large-scale development, integration, validation, maintenance and is not effective in reusing common components or addressing redundancy of solution elements. Emergent design alone is also associated with decline is system qualities/non-functional requirements, delays or reduced velocity due to excessive redesigns and a reduction in synchronisation and collaboration. On the other hand, the framework recognises that traditional IT-architecture approaches (intentional architecture alone) extensively led to early architectural (re)work, with unvalidated designs, abundant documentation and rework if new requirements emerge or old requirements change. Thus, the combination of intentional architecture and emergent design in the right proportions help to address the complexity of building enterprise solutions, as it supports current user needs while allowing evolution of the system to meet future needs in a complex environment (Scaled Agile, Inc., 2021).

## 4.3. An initial theoretical framework devised of three conceptual interaction models

What is interesting that all these aspects of both working practices can be enhanced by governance, albeit in a 'make or break' type relation, as governance determines, roles and positions, rights, responsibilities and authority as well as monitoring and control systems. All these aspects could potentially influence the balance points, i.e., a solution architect that is monitored and controlled on delivered designs might not have the right incentives for multiple extensive face-to-face meetings with developers. My initial theory, captured in Figure 7, was that governance strategies that block or disturb the creation of the needed balances also disturb the complementary nature of the combination or could even create the opposite effect: problems that are worse than if one of the practices was badly governed on its own. While the right governance strategies can enhance the complementary nature of both software development practices by enabling stakeholder to obtain and maintain the right balance points by defining the right roles, responsibilities, incentives etc. Based on this theory the following high-level conceptual model of architecture-agile interaction was devised, resulting in three interaction models:

1) IT-architecture is dominant;
2) Development with a balanced exchange model between IT-architecture and agile software development; and finally
3) agile is dominant.

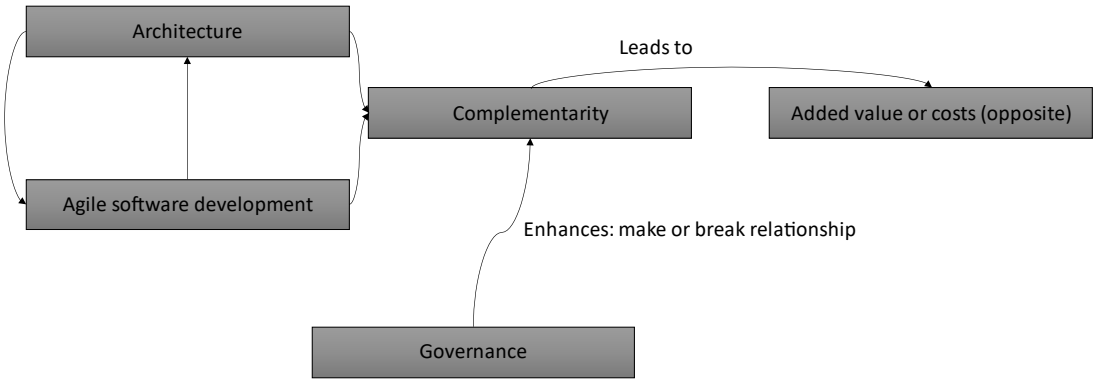


*Figure 7: Initial conceptual model of theory*

While these classifications are extremes, extremes can be useful to elicit underlying behaviours that are not observable in more nuanced situations[48] and can serve as points of reference (Flyvbjerg, 2006). Phenomena from extreme cases can be generalised to more nuanced cases by means of analytical generalisation, for example, if a phenomenon unexpectedly occurs under extreme circumstances this can be proof that the phenomenon can occur under less extreme circumstances. Similarly, if a phenomenon does not occur as expected under the 'perfect conditions' then this can be evidence of why a phenomenon does not occur in more nuanced cases.

The discussed similarities and differences were then compared to findings of the case studies and the characteristics of the agile and architecture dominant classifications have been defined. An overview of these characteristics is presented in Table 7. The balanced exchange model can share characteristics of the other classifications. No specific criteria for the balanced exchange model were defined, as there were no practical examples of this before this study. Not all characteristics had to be met in a development process to be classified as a certain type. As, the balance point has been found to differ for each application in literature.

*Table 7: Characteristics of architecture and agile dominance classifications*

| **IT-architecture dominance characteristics** | **agile dominance characteristics** |
|---|---|
| Directionally composed by IT-architects | Development team stakeholders are ones that compose the software/solution architecture |
| Little/no alterations possible after composition of IT-architecture | Frequent alterations to IT-architecture based on new insights or issues encountered during development process |
| Development team initially not considered in composition of IT-architecture | IT-architects outside the development team initially not considered in composition of IT-architecture |
| Layered architecture team structure | No formalised software/solution architect role, self-organisation to reach software/solution architecture or lack of recognition for IT-architecture by development team |
| Interaction with development team is upon transfer, in case of problems and upon delivery of the project only | Face-to-face interaction had a central role |
| Since not everything can be known beforehand, issues for the development team arise due to the IT-architecture and/or it's inflexibility | Issues arose due to a lack of upfront planning or architecture |
| Waterfall or hierarchical approach | Short cyclical approach |
| Uncertainty is addressed up-front | Uncertainty is addressed on the go |

[48] For example, because they are crowded out by the sheer number of behaviours.

Characteristics of an IT-architecture dominant software development process are directionally composed IT-architectures by IT-architects; IT-architectures that are difficult to alter based on new insights by developers after composition; development teams' perspective not being considered while composing the IT-architecture; layered architecture team structure; interaction with the development team at limited, pre-specified moments such as transfer of the IT-architecture, problems or delivery of the project; problems arise due to the IT-architecture or its inflexibility, as not everything can be known beforehand; uncertainty is address up with up-front development; and the approach to software development is hierarchical or waterfall.

An agile dominant software development process could be classified by: development team stakeholders are the ones composing the software/solution architecture; frequent alterations to the IT-architecture based on new insights or issues that were encountered in the development process; no formalisation of the software/solution architecture role or a lack of recognition for IT-architecture by the development team; issues that arose due to a lack of up-front planning or architecture; uncertainty is addressed on the go by the development teams; face-to-face interaction has a central role; and a short-cyclical approach is used. A case study with more initial architecture than needed would fall into my architecture dominant category, while a case study with more emergent architecture than needed would fall into the agile dominant category. Case studies with a balanced exchange model enjoy the best of both worlds, they are able to plan for foreseen future issues and use their agile process effectively to deal with unforeseen issues.

## 4.4. Conclusion of chapter 4

The very aspects that can lead to an interpretation of architecture and agile development as counterparts can also be the theoretical foundation as of why architecture and agile software development have potential to be complementary. The similarities of agile and IT-architecture roles are not present in the approach, but in the deliverables, as they both strive to create working software of good quality.

1) Both up-front as well as emergent architecture design can be useful, depending on the situation.
2) Functional requirements add business value to stakeholders, but need non-functional requirements to support these functional requirements.
3) There is an opportunity to balance the threats associated with both long-term waterfall working methodology as well as the threats associated with short-cyclical iterative working methodology. As, both approaches can take their way of working too far.
4) Documentation is needed to communicate when people are unavailable, but cannot replace face-to-face interaction.

The SAFe framework proscribes a balanced exchanged model, that balances up-front architecture with emergent architecture. This balance point differentiates over time in the development process and the complexity of the environment. This balance has to be managed by the solution architect, which is a new role located in between the traditional enterprise and software architecture roles.

The following three conceptual interaction models have been formulated.

1) IT-architecture is dominant;
2) Development with a balanced exchange model between IT-architecture and agile software development; and finally
3) agile is dominant.

IT-architecture dominant interaction models uses more up-front planning than is needed to address foreseeable problems, while the agile dominant interaction model encounters problems through their agile process that they could have foreseen with up-front planning. The balanced exchange model uses both up-front planning for foreseeable problems and their agile process for unforeseeable issues effectively.

# 5. Empirical IT-architecture and agile software development interactions

This chapter will provide rationale for the case study selection and will discuss how the characteristics of each case are related to the selection criteria. Additionally, the cases are characterised as green or brown field situations. Then the selection of interviewees is discussed. Finally, each case is classified in an interaction model as defined in chapter 4.2.

## 5.1. Case study selection and characteristics

First some general remarks for selection criteria are discussed. Then cases are discussed case by case. Remember that selection was informed, trying to select cases such that agile and architecture dominant cases, as well as a balanced exchange model would be found. The expectations for each case are discussed in the case-by-case section. Table 8 shows which selection criteria each case study met.

*Table 8: Adherence of case studies to selection criteria*

| Criterium | Case 0 | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 | Case 6 |
|---|---|---|---|---|---|---|---|
| Case took place between 2016 and 2021 | | x | x | x | x | Development phase, requirements phase earlier | x |
| Project should involve an agile development team and at least one architecture role. | x | | x | x | x | x | x |
| Organisation type | x | x | x | x | x | x | |
| Geographical scope | Central, outsourced development on site | Central, outsourced development on site | Mixed, outsourced development off and on site | Central, inhouse development | Central, inhouse development | Central, inhouse development | Decentral, development outsourced overseas |
| Development process up to a specific release or development process up to transfer to operational phase | x | x | x | x | x | x | x |
| Specific agile framework | Scrum | None | Scrum | Scrum, elements from SAFe | SAFe | LeSS, currently SAFe | SAFe |
| Access to expertise | Only agile perspective | Both perspectives | Both perspectives | Both perspectives | Only architecture perspective | Only architecture perspective | Both perspectives |

### Organisation type and access to expertise

A total of 12 interviews has been administered with various roles over 7 distinct organisations. After having interviewed six public organisations, the decision was made to collect data on a private organisation, instead of a new public organisation, as saturation had been reached on new information for the public sector. This is case study 6. Thus, two of these interviews have been conducted at one private organisation to shed light on the degree to which the findings were related to the government context or are relevant in a broader context. For all cases it was possible to interview both agile and IT-architecture roles. For case study 0 only an agile role was available, for case studies 4 and 5 only IT-architecture perspectives were available.

### Actuality

The time criterium did not did leave older cases unconsidered, as shown in table 8. Case studies 0 was considered even though it did not adhere to the selection criterium of taking place in the past five years. The reason that this case study has been included is that their development practices considering the interaction were novel for the time they were carried out in. Case study was also included, stakeholders needed over ten years to reach agreement on requirements, as the software application needed to function in a network of heterogeneous, European stakeholders. However, the development process did take place within the past five years. All other case studies adhered this criterium fully.

### Agile and architecture frameworks

After several interviews it became apparent that the NORA architecture framework is binding for government organisations. It seemed that IT-architecture frameworks tend to play a different role than agile frameworks (NORA, 2019). As the NORA is prescribing on solutions and processes, while agile frameworks are focused on how to organise processes. All other cases did adhere to this selection criterium. Most case studies implemented architecture with the TOGAF frame in mind as well, case study 1 is an exception to this.

### Geographical scope

While all cases were developing centrally, except case study 6, which outsourced their development oversees, their organisation models differed. Case studies 3, 4 and 5 all developed software inhouse using their own teams. These and organisation 6 were larger organisations, having employed several hundreds of people if not more. The smaller organisations outsourced their development within the same country and worked on site, except case study 2. In case study 0, 1 and 2, the client was a large public entity, that delegated (part of) their responsibility to one or several smaller entities, resulting in a collaboration of private and public parties.

### Case by case description

Case 0 was the pilot interview, which has been added as the pilot was conducted with an actual practitioner and contained useful data. Agile software development was very new at this time in the Dutch public sector. The project worked agile stealthily at the start, as the agile methodology was not very well understood by the program manager overseeing the programme. This meant that the team worked according to the Scrum methodology, but phrased the concepts in project management language. Over the course of the development process, agile concepts were gradually introduced and well received.

Case study 1 did not follow an agile framework, while it did work agile, using sprints, retrospectives, a backlog, product owner and refinements. One of the participants remarked that this was how they were used to work. The case study was characterised by a crisis structure as well as a green field situation and thus was expected to be agile dominant, as there was more adaptability required due to uncertainty and standardisation is less important. This case study was the only case study that had no formalised architecture role.

Case study 2 did not follow the SAFe framework, as the project was too small to implement the framework effectively. Consequently, they employed Scrum, which is suitable for projects with a limited number of teams. It was expected to be agile dominant, since the organisation and team were relatively small.

Case studies 3, 4 and 5 were expected to be architecture dominant due to the organisational size, which tends to invoke more compliance and standards to adhere to. Case study 3 did follow the Scrum framework, while cherry-picking elements from SAFe. This case study experienced problems with their context as this was not suited for agile according to participants. Case study 4 did meet all the selection criteria, however the agile perspective role that had committed to the interview became unavailable and a replacement was not found. Case study 5 could only provide the IT-architecture perspective as well. This project used a lightweight version of the LeSS framework. But missed coordination among

different agile teams and as a consequence the organisation has recently started implementing SAFe. Both case study 4 and 5 have agile development processes as non-agile, more traditional development processes, which allowed for interesting comparisons during the interviews.

Case study 6 is a private sector organisation that was added to identify whether problems with a non-agile context were specific for the government sector or not. The interaction model is interesting as software development is outsourced to another country and time zone. All the case studies were characterised by complexities in (external) stakeholder dependencies.

## Green vs brown field case studies

As the amount of existing systems, standards, dependencies etc. can affect the need for agility or architecture, Table 9 shows whether a case is a green or brown field situation. A green field operation is building a completely new system, or a new type of system for the organisation. While a brownfield situation could be an updated version of a system, a new system that has to be integrated with (many) existing systems. The green or brown field situation was also linked to knowledge and uncertainty, as green field situations often require new knowledge and tend to be more uncertain than brown field situations. Thus, one would expect IT-architecture dominance in green field situations and more agile dominance in brown field situations. Case study 6 has not been classified, as in this case study a very high-level discussion was held on a very large software platform that included several teams and applications. It was unclear whether this is a green or brown field situation based on the interviews, as the interviews were more focused on checking certain statements made by other participants to see if they held out in a private sector organisation.

*Table 9: Green or brown field situation for each case*

| | **Green field** | **Brown field** |
|---|---|---|
| **Case study 0** | x | |
| **Case study 1** | x | |
| **Case study 2** | | x |
| **Case study 3** | x | |
| **Case study 4** | | x |
| **Case study 5** | x | |

### Green field

Case study 0 was classified as a green field situation as their agile approach was new at the time. Part of the teams worked in a green field technology situation as well, having no prior systems, while some teams did have an old system and its requirements to look at. Thus, the case shared elements of a brown field situation in this perspective, but was not fully brown field either. The system for case study 1 was entirely new for the organisations tasked with its development, consequently it has been classified as a green field situation. Participants in case study 3 stated that the application they were discussing was entirely new for the organisation, resulting in a green field classification. Finally, case study 5 was also classified as a green field situation as there existed no previous network that connected applications for this purpose, stakeholders had to define their own requirements and reach agreement upon those.

**Brown field**

Case study 2 considered a migration from one platform service to another, the requirements were largely the same. Thus, it was a brown field situation. Case study 4 considered a modernisation of services, resulting in a brown field classification as well.

## 5.2. Selection of interviewees

This section gives further details on the cases, like the roles and experience of participants involved. The implementation of these and other relevant roles are also shortly discussed. Table 10 shows which roles have been interviewed for each case, as well as their experience.

*Table 10: Roles interviewed for each case study*

| Case study no. | Role 1 | Experience in role | Role 2 | Experience in role |
|---|---|---|---|---|
| Case study 0 | Scrum master/functional developer | < 5 years | n.a. | n.a. |
| Case study 1 | Back-end development team lead/Solution architect | 5-10 years, first time in political playing field | Back-end developer | < 5 years |
| Case study 2 | Back-end development team lead/Solution architect | 10 – 15 years | Back-end developer | < 5 years |
| Case study 3 | Enterprise architect | 10 – 15 years | Product owner | 5-10 years |
| Case study 4 | Enterprise architect | 10 – 15 years | n.a. | n.a. |
| Case study 5 | Domain architect | 15 – 20 years | Enterprise architect | 15 – 20 years |
| Case study 6 | Domain lead | 10 – 15 years | Product manager | 5 – 10 years |

The pilot interview was conducted with a professional that shared a Scrum master and functional developer role. These roles where combined for the majority of the project and later split up. The interviewee then assumed the functional developer role and passed on the Scrum master role to somebody else. The interviewee had < 5 years of experience as a functional designer, the Scrum master role was completely new as agile was a new methodology at this time. An solution architecture role was approached, but unavailable for an interview, so only the agile perspective was represented in this case. However, there were enough IT-architecture roles consulted in the other interviews.

For case studies 1 and 2 a back-end developer and the team lead of this back-end development team were interviewed. In both cases the tech lead was recognised as a solution architecture role as well. Thus, both cases contain both perspectives. The back-end teams worked together with front-end teams in synchronised sprints in the two cases. Both the back-end and front-end teams were outsourced from different organisations in both cases as well. In the first case study the solution architect/team lead role had 5 – 10 years of experience, however it was their first time in a project that involved a lot of politics. The solution architect/team lead role had 10 – 15 years of experience. The back-end developers had <5 years of experience in both cases.

Case study 3 consisted of an interview with an enterprise architect with 10-15 years of experience, as well as a product owner with 5 – 10 years of experience. These roles added new perspectives, as these were the first enterprise architect and product owner roles that were interviewed. This enterprise architect was also the lead of the IT-architecture team. The enterprise architect role typically had less interaction with developers and more interactions with other architects, solution architects in this case. In general, enterprise architects were found to have interaction with more senior stakeholders, internally in their own organisation, as well as externally with stakeholders from other organisations, for example concerning external dependencies of systems. The product owner role has the responsibility to deliver the software solution and determines priorities for/with the development

team.

In case study 4, an enterprise architect as well as an IT-lead role were available for interviews. However, the IT-lead role became unavailable. The enterprise architect had 10 – 15 years of experience. In case study 5, two architecture roles were available for interviews: an enterprise and domain architect. Both roles had 15 – 20 years of experience. It was tried to approach an agile development team role as well, however unsuccessfully. The enterprise architect in this was also the lead architect and organised their architecture team according to the TOGAF model: having an enterprise, business, technology architects. As well as domain and solution architects. The technology architects were specialised in a specific technology, while a domain architect covered a specific domain within the organisation and formed a bridge between various IT-architecture roles. For example by helping a solution architect to fit their solution under the domain and enterprise architecture. Thus, case study 5 investigated an interesting combination between the TOGAF and the SAFe frameworks.

In case study 6, a product manager and domain lead were interviewed. The product manager role oversaw and coached various product owners that were working on a platform that connected, business clients, consumers and suppliers. The product manager role had 5 – 10 years of experience and has had a background as developer. The domain lead role had 10 - 15 years of experience. It seemed to be comparable to a lead architect or domain architect role. As the domain lead had multiple delivery managers, solution managers and solution architects working in their team and oversaw the business to consumer side of things. The solution architects were responsible to work across various teams and integrate several different technologies, as development teams were very technology driven.

## 5.3. Classifying the interaction model of each case study

The research approach to address the empirical perspective of the first research question was to first colour code transcripts and put relevant data in a tabular case study report Appendices C-I. Then to characterise each case study in the conceptual model of the previous section.

### 5.3.1. Architecture dominant case studies

A final overview of which architecture dominance characteristics cases exhibited is given in Table 11. Note that case study 1 did not have any of the architecture dominance characteristics.

*Table 11: Case studies mapped to architecture dominance characteristics*

| architecture dominance characteristics | Directionally composed architecture by architects | Little/no alterations possible after composition of architecture | Development team initially not considered in composition of architecture | Interaction with development team is upon transfer, in case of problems and upon delivery of the project only | Issues for the development team arise due to the architecture and/or it's inflexibility | Since not everything can be known beforehand, issues for the development team arise due to the IT-architecture and/or it's inflexibility | Uncertainty is addressed up-front | Waterfall or hierarchical approach |
|---|---|---|---|---|---|---|---|---|
| **Case study 0** | | | | | x | | | x |
| **Case study 1** | | | | | | | | |
| **Case study 2** | | | x | | | | x | x |
| **Case study 3** | x | | x | | x | x | x | x |
| **Case study 4** | x | | x | | x | | | x |
| **Case study 5** | x | | x | | x | x | x | x |
| **Case study 6** | | | | | x | | x | x |

In cases 4 and 5 the process flowed in a waterfall like way, as depicted in Figure 8, with an agile process only at the bottom in formulation of user stories and their translation into working software. Because of this, these software development processes were classified as IT-architecture dominant cases. The process usually started with either a minister announcing something, for example an implementation of a new law or with an internal business stakeholder that had an idea, such as a new software application that optimised the workflow in a business process. For now, the important notion is that this interaction model is vastly different from what the SAFe framework, that organisations wish/try to adhere to, propagates (Figure 9). This approach created fragmentation and throw it over the fence behaviour instead of multi-disciplinary teams that take co-ownership and co-create solutions that align business processes with the IT landscape. While there was some multi-disciplinarity in the teams, by mixing architects and business stakeholders and sometimes product owners, development team roles were not considered at the start on a structural basis.

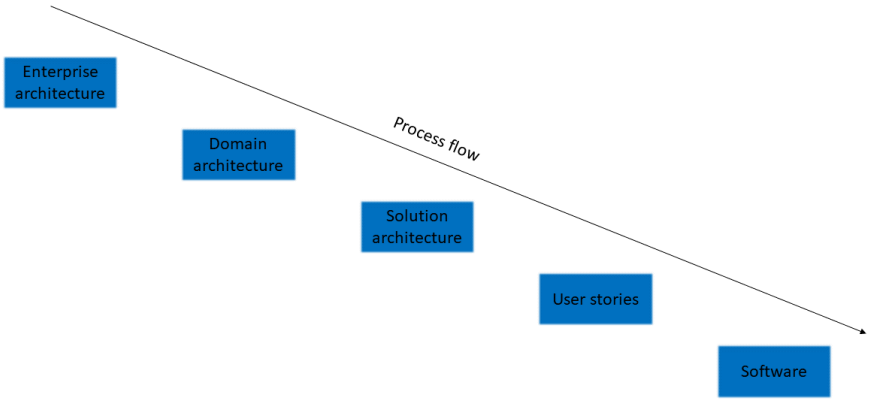


*Figure 8: Waterfall process flow*

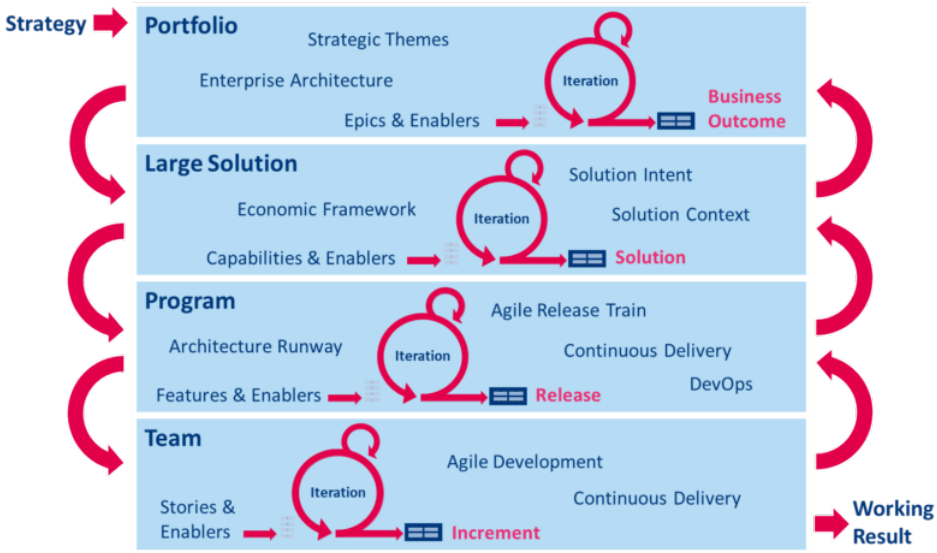


*Figure 9: SAFe agile architecture process flow (Lankhorst, 2019)*

The waterfall model, depicted in figure 8, usually followed a similar structure, an enterprise architect discussed with external and senior internal stakeholders how the new idea or law would affect and fit into the current enterprise landscape. This concerned the business processes as well as IT. Cooperation with developers was not present or minimal, I.e., quarterly. In all case studies interaction between the enterprise architect and developers was minimal. For example, in case study 3 the enterprise architect(s) presented the new architecture, consisting of principles, frames/guardrails and guidelines, and the development teams could listen and ask questions. In case studies 2, 4 and 5 there was no face-to-face interaction between enterprise architects and developers, as this was the task of domain and/or solution architects. After the enterprise architect was finished finding agreement on requirements with external and senior internal stakeholders, the relay baton was passed on to the domain architect or solution architect, depending on the size of the organisation. A domain architect was responsible for a specific domain of the enterprise in a larger organisation and thus implements the changes of the enterprise architecture into their domain architecture. In this process the architecture lost some of abstraction and became more fine-grained. This continued as the domain architect passes the changes in the domain architecture down to the solution architect, which translated the domain architecture into a solution architecture, Project Start Architecture (PSA), technical design, list of requirements or a combination of these. Based on the PSA, technical design, requirements document or solution architecture the development team started to formulate user stories together with the business. The deliverable of the solution architect was in practice anything between a predefined blueprint ready for implementation to a set of guidelines, principles and frames.[49] Thus, there was a varying amount of freedom for the development team to work with, even for distinct development processes within organisations, depending on the solution architect. These user stories are then transformed into working software through an agile process, meaning that in essence the agile process only took place at the end of the process.

There was a clear reason why architects are usually the role to affect a software development process[50] first in a government context. First, they had technical expertise as well as access to senior stakeholders that was needed to start with the translation of a legal text into a technical solution. The architects worked out a high-level architecture for the chain, or network of stakeholders and then how this affected their own enterprise architecture landscape, which was then translated into a Project Start Arichitecture (PSA), which could range from a high-level description of the solution to an extensive document with hundreds of requirements or even snippets of code. Across and within organisations there seemed to be no general definition on what should be included in a PSA to frustration of both IT-architects and developers. Second, as this process was formalised through the NORA and in such a way that allowed for discretionary freedom of experts (ICTU, 2021b). This often led to complaints of developers that they received either too little frames to work in or were that too much was predetermined. This confirmed the findings of the literature review that the amount of architecture that is needed is a case specific balance point.

While initially, the role of a software architect was thought to be found present closely to agile development teams, this role was not found. Instead, solution architect was the architect that was closest to the development team. Even in cases where the SAFe framework was not used at the time of the development process. Thus, it might also be that the participants perspectives are enriched with their current knowledge when reflecting on past software development processes. Even though the framework prescribed the use of a system architect, which is closer to a technology, application or software architect role (Scaled Agile Inc., 2021). Case studies 5 and 6 specifically stated that they used a technology architect role, while an enterprise architect in case study 3 mentioned specifically that this role was lacking within their organisation. In case studies 0 and 2 an infrastructure team or DevOps and Implementation Engineer roles were comparable to the software architecture role.

[49] Constraints or must have requirements.

[50] Often called a project, programme or procedures, but usually has project management like characteristics in terms of governance and management.

The way organisations coped with feedback on the architecture from the development team differed greatly across and within organisations, again depending mainly on the architect. This was an important factor in the classification of the conceptual interaction model. Where some architects sought to balance short term functionality with long term sustainability of the IT landscape more according to interactive theoretical model in Figure 9, others dug themselves into trenches, defending their architecture, sticking to the theoretical model of Figure 8. Not since they did not want to cooperate, but since the proposed changes were only helpful in the short-term and would cause issues in the long term. The degree to which architects allowed changes to the architecture varied. For example, in case study 4, deviations from the architecture were allowed, but needed to be cleaned up after several months in such a way that the operation becomes unattended and unmonitored. A similar formalised process exists in case studies 5, however with limited success.

### 5.3.2. Agile dominant case studies

Case studies 1 and 3 were classified as agile dominant. Table 12 presents an overview of which cases have which agile dominance characteristics.

*Table 12: Case studies mapped to agile dominance characteristics*

| agile dominance characteristics | Development team stakeholders are ones that compose the architecture | Frequent alterations to architecture based on new insights or issues from development process | Solution / software architects outside the development team initially not considered in composition of architecture | No formalised architect role | Lack of recognition for architecture by development team or architecture is not considered | Issues arose due to a lack of upfront planning or architecture | Uncertainty is addressed on the go | Short cyclical-approach used in solution architecture design. | Face-to-face interaction had a central role in establishment of software/solution architecture and agile software development |
|---|---|---|---|---|---|---|---|---|---|
| **Case study 0** | | x | | | | | x | x | x |
| **Case study 1** | x | x | x | x | | x | | x | x |
| **Case study 2** | x | | x | | | | | x | x |
| **Case study 3** | | | | | x | x | x | | |
| **Case study 4** | | | | | x | | | | |
| **Case study 5** | | | | | x | x | | | |
| **Case study 6** | | | | | | x | x | x | x |

In case study 1, the software/solution architecture role was not really formalised and recognised. Still, both interviewees in this case study pointed towards the same person when asked who fulfilled this role. This was one of the cases where the Team lead was also envisaged as the solution architect. Interaction surrounding architecture issues was frequent and formalised as well as initiated outside formalised meetings by both parties. The solution architecture foundation was laid in the first week and further developed 'along the way.' For example, the Team lead/solution architect would address issues with tooling that the development team experienced. The responsibility for the solution architecture was clearly with the development team here. Solution architecture items were put on the backlog and implemented through sprints, just as in other case studies. Thus, this could be an example of emergent architecture dominant case as described by the SAFe framework and was classified as an agile dominance case in the basic typology of interaction models.

Consequently, case study 1 was an example of just starting to build things, as a minimal solution architecture has been developed by the person who started the project. Which was up and running after

a week. While there are some architectural designs, like a drawing. This seemed limited when compared to other cases, where the designs had passed the hands of various architecture roles before it reaches the development team. Further evidence was a description of how technical problems emerged during the project, the question is whether a formalised architecture role could have identified these problems before hand, for example when planning a few sprints ahead, and advised the team how to cope with quality problems. Which was something case study 2 employed. They planned a few sprints ahead with the development team, PO, operations, solution architect/team lead and identify issues that may cause problem and assign these to be researched by the relevant stakeholder. This way they cleared as much impediments they were able to foresee as possible and could be an explanation for their lack of technical debt.

The solution architect worked out the solution architecture together with the lead developer in case studies 3. However, this did not go as intended, as trouble with ownership was encountered. In case study 3 the IT-architects had their own informal procedure to address these differences of opinion on working under the IT-architecture. In case study 3, the enterprise architect escalated the decision to ignore the IT-architecture to management, as the consequences were their decision. They wrote a scenario document for both following and ignoring the architecture and presented these to the management. Usually, the relevant architecture was upheld, but not always. There were several examples given by the enterprise architect of how their opinion was ignored and caused issues later (Appendix F).

Since the developers built working software and pushed this into production, while IT-architects had an advisory role, the informal decision-making capacity was with the developers in case studies 3 and 5. Even though there were formal escalation procedures to management, where IT-architects sometimes got the formal support of management, this did not lead to developers doing what the architects want in case study 5. As reworking to make the application fall under the architecture again had low priority with management, PO's and teams while human resources were scarce and demand for (alterations in) applications was high. Resulting in the changes standing on the backlog indefinitely. Development teams seemed to have become aware that they could ignore the IT-architecture without real consequences and started regarding the IT-architecture as some documentation that nobody read or should read, often with large rework phases and a lot of technical debt on the backlog as a result. Sometimes even blaming the IT-architect team for the rework and technical debt.

While other cases did have characteristics of an agile dominant interaction model, they were not classified as such. All case studies used an short-cyclical approach for their software development, however case studies 4 and 5 had a mix of both traditional and agile development teams.

### 5.3.3. Balanced exchange model case studies

In case study 2, the solution architect and team lead were consolidated into the same role. The solution architect thought out an unfinished blueprint. Interestingly, their interaction model came close to what the SAFe framework prescribes. A blueprint with holes in it was used as a metaphor in the interview. Physical workshops were organised specifically to fill these holes over the course of the development process, together with other relevant stakeholders, for example from operations, to ensure sustainability of the solutions. Hence this case studies shared characteristics of agile and IT-architecture dominance models. Development team stakeholders were not initially considered, but were involved for specific issues during special architecture workshops. This case study experienced almost no technical debt and interviewees reflected positively on the project.

Case study 0 did not take place in the past five years, which is relevant, as it had an interaction model that was similar to that described in SAFe, even though SAFe did not yet exist at the time of the project. Consequently, it was classified as having an balanced exchange interaction model. In this software development process, the PO and Scrum master/functional developer were involved in the formulation of the requirements in the PSA. Requirements from the PSA were then translated into features using the refinements, up to two sprints ahead. During this process various stakeholders were present, a solution architect was consulted before and during the refinement process. The solution

architect helped to represent the non-functional requirements, while the PO represented the functional requirements. In addition, a Project End Architecture (PEA) was made by solution architects at the end of the development process. This documented all new insights and developments that defined the new solution architecture state and was then transferred to the administration organisation.[51]. If this happened in an iterative process, it came close to an agile architecture. However, an agile architecture is developed in an incremental way up-front and during the development process to assist development teams, while the PEA was mainly for operations.

What is interesting about case study 5 is that it ran agile software development processes as well as waterfall software development projects. Consequently, not every team had an assigned solution architect. This organisation used an incremental architecture on an agile development process, this incremental architecture process was called an MVA, minimum viable architecture. However, since the discussed project was an older project, the non-iterative PSA procedure was used. The domain architect that worked on the project tried to keep some room in the PSA for the development team and solution architect to work with. In doing so, he experienced pushback from colleague architects. For now, it is important, that in this case study, during the specific software development process that was discussed there was an exchange model present. However, it is difficult to say how interactive this exchange model actually was, as the solution architect made the first iteration of the MVA with designers. The later iterations of the MVA were delivered in dialogue with the development teams.

In case study 6, the architecture was designed in an iterative, agile process. The solution architects had a critical position that looked across teams, as individual teams are very technology driven. Meaning they were specialised in a certain technology, such as a programming language, but unable to identify the effects of their work on other parts of a solution that used a different technology. The solution architect worked together with development team leads, designers, product owners, senior testers and a business analyst in a 'horizontal bubble.' A first design of the solution was made and challenged by the other stakeholders, adding insights that the solution architect had missed. The amount of involvement of the solution architect was determined by the uncertainty that a capability had. In general, for completely new or big impact capabilities the solution architect was more involved than in smaller, more procedural changes. However, in the latter case the architect was still involved, albeit more in a reviewing role, for example in refinement and end-to-end testing.

A final overview of the classification of cases to interaction models in the typology is given in Table 13. It would be interesting to be able to make more distinctions within balanced exchange model, as in this model, case studies seemed to differ more than across cases than in architecture dominant cases. Case studies 1, 3, 4 and 5 were classified according to expectations of the case study upon selection. Case studies 2 and 4 differed from the expectation.

*Table 13: Initial classification of case studies to interaction models in typology*

| Interaction model | Agile dominant | Balanced exchange model | Architecture dominant |
|---|---|---|---|
| **Case 0** | | x | |
| **Case 1** | x | | |
| **Case 2** | | x | |
| **Case 3** | x | | |
| **Case 4** | | | x |
| **Case 5** | | x | x |
| **Case 6** | | x | |

[51] Operations

## 5.4. Conclusion of chapter 5

Characteristics of the selected case studies have been discussed. The roles of the interviewees in the case studies were also presented. Then the case studies were classified using the typology that was presented in chapter 4. Two case studies has been classified as agile dominant, two case studies as It-architecture dominant and four case studies have been classified as a balanced exchange model. One case study had both waterfall development processes as agile development processes in the organisation. Two cases were not classified according to expectations upon selection. For the balanced exchange models it was difficult to determine how they were balanced with the typology of chapter 4. It would be interesting to be able to make more distinctions within balanced exchange model, as in this model, case studies seemed to differ more than across cases than in IT-architecture dominant cases.

# 6. The influence of context factors on empirical interaction models: an extended typology

This section discusses how context factors impacted the empirically identified agile-architecture interactions. Context factors from literature where validated. New context factors have been found: trust and stability.

## 6.1. Communication, trust, stability, knowledge and perceptions

Since communication, knowledge and the software architect and team have been found as important factors for IT-architecture and factors that are related to governance, these factors were discussed with participants to start a discussion on how governance affected the interaction and added value. Expected perceptions between the role of IT-architect and of agile developer have been identified in some cases, while in others they were absent. Trust and stability are new factors that were included due their occurrence across cases.

### Communication

In case study 3 and 5 there was mention of internal conflict between several groups. For example, between solution architects and developers, but also among IT-architects and developers themselves. One explanation could be that these organisations are more large scale and that people that interact on a team-to-team basis have less incentives to avoid confrontation than people who work in small teams. However, the organisation of case study 3 is smaller than the organisation of case study 5. Another explanation could be that people or teams interact less often in these organisations. However, there was also mention of conflict within certain roles, it is unclear whether this was always between different teams sharing the same role (I.e., developer or domain architect). In both cases it was clear that it concerned different teams. Participants argued that a certain degree of conflict or difference in perspective is needed to build something that works for all stakeholders. On the other hand, too, much conflict can be destructive as it can lead to people not willing to interact anymore. These are clear examples of how communication affects the interaction models.

### Trust and stability

While the literature review identified risk management, communication, the team. Software architect and knowledge as important factors that influenced the combination, interviewees in case studies 0 and 1 emphasized the role of trust when asked for benefits, while case studies 1 and 2 linked trust to the stability of the team. In case study 1 mainly in the negative sense, seeing team members from other teams leave due to a lack of trust. In case study 2 more in the positive sense, allowing to create a stable sprint rhythm through trust.

### Knowledge

Trust can be increased by knowledge in each other's practices or decreased by ignorance on each other's practices. The disregard of It-architecture in some cases by teams were examples of a lack of architectural knowledge that led to technical debt. While the inability of IT-architects in other cases to alter their designs on new insights by the development teams was an example of a lack of knowledge on agile methodologies. One participant in case study 5 made an interesting statement on how they envisioned their organisation in a perfect world: this was an organisation that was very experienced and knowledgeable in agile and IT-architecture, where architecture knowledge would be an inclusive part of the agile development teams, beyond the function of a solution architect, with team members taking on the role of an solution architect. However, this would require organisational development, as this would require developers with It-architecture knowledge. A similar vision was expressed in case studies 3 and 6.

### Perceptions

The perception of both participants in case study 3 was that IT-architecture lays the foundation and agile could help to design the building on top of the foundation. And both can work together to reach a good end product. If you do not lay this foundation with IT-architecture, a team will take shortcuts that lead to issues later on. This perception is interesting as it is quite IT-architecture dominant. Such an perspective, shared by both the agile and IT-architecture perspective are very likely to have influenced the interaction model in this case. Two interviewed IT-architects from case studies 3 and 5 were unable to interact with a development team member, due to the simple fact that they were an IT-architect and this development team member had an unfavourable perception of IT-architects. These are empirical illustrations how perspectives on both roles affect a software development process.

## 6.2. Uncertainty and risk

First the definitions of uncertainty and risk are given to illustrate the difference, then their influence in case studies is discussed.

### Uncertainty

Uncertainty are white spots in knowledge or future events. Often expressed in quantitative distributions, i.e. percentages of occurrence, or qualitative ordinal rankings, such as low, medium, high classifications.

### Risk

Risk is uncertainty multiplied by impact. Thus taking into account the consequences of the possible outcomes as well. This could also be a quantified measure or a more qualitative ordinal scale.

### Risk management

The practice of identifying and mitigating risks through the use of governance strategies that investigate or mitigate the uncertainty or impact of a risk.

### Occurrence in cases

The main difference between uncertainty and risk is that uncertainty does not consider the impact yet. Risk management is important for us as Waterman (20181; 2018b) introduced risk tolerance of an organisation as a way to determine how up-front architecting should be balanced with agile architecting. The following uncertainties and associated risks have been found in case studies:

- Staffing: finding the right people.
- Requirements: will requirements change? Will new requirements come up in the development process? What will be the impact of this?
- Technology: legacy systems and technical debt. How will this affect the project?

In case study 5 the IT-architects tried to make their designs waterproof and deliver them ‘finished.’ However, since new things kept popping up, the designs were never actually finished. Even though the deadlines of the development teams were fixed. The IT-architects intention was not to hamper the developers in their progress, but to manage risks and uncertainties. The agile methodology was founded to cope iteratively with a specific kind of uncertainty. Empirically, it has been found that rework and technical debt could be a consequence of up-front architecture as well as the short-cyclical agile methodology. Since not everything can be known beforehand, up-front architecture design have been found to be a cause of technical debt and rework in case study 4. IT-architects were not always able to assess which issues developers would encounter, thus the IT-architecture could not consider these issues. Similarly, if the IT-architect has been overly fearful of certain issues, that had only limited effect on the development process (or other processes). This led to overly stringent requirements or frames for developers to work with. The other way around, just starting to build something in a short-cyclical way

led to developers running into issues that could have been simply avoided with more up-front planning and coordination on when to build what. This has been found in case study 1. Thus, up-front architecture could reduce known uncertainties and risks, while the agile methodology could help to tackle unknown uncertainties risks.

Case study 0 addressed uncertainty on requirements by using refinement and demo[52] sessions, which is interesting as another case study, number 1 and 3, struggled with getting these sessions right and experienced problems due to too much uncertainty on requirements. In case study 2 participants also stated the stories and agile approach helped them to address uncertainty. Interestingly, case study 6 followed Waterman's (2018a & 2018b) theory on managing uncertainty in an economic sense. One participant defined two extremes to address uncertainty and associated risk: 1) up-front planning, so dependency and risk management or the 'normal' project management approach if impact or uncertainty are large; 2) if the uncertainty or impacts of the risks are low: proceed in an 100% agile way to incrementally build and solve impediments. The participant concluded by stating that there is a combination of both. Thus showing that Waterman's (2018a & 2018b) theory is implemented in practice.

## 6.3. Improving the typology

The balanced exchange model can be split up based on these context factors. Four new interaction models are introduced with their own characteristics:

1) The carry over or ping-pong model;
2) The louse in pelt model;
3) The solution architect as cooperating foreman; and
4) The co-development model.

Note that the first model leans more towards the IT-architecture dominance model, while the fourth model leans closer to the agile dominance model. Consequently, these four interaction models are archetypes for balance points of IT-architecture and agile in software development. Both the IT-architecture and agile dominant interaction models can considered as more extreme models than before, due to the new nuances in the balanced exchange models. The individual models and their characteristics will be discussed next.

### 6.3.1. The carry over or ping-pong model

The most IT-architecture dominant exchange model is the carry over or ping-pong model. It has the following characteristics:

1) IT-Architects starts with up-front design and throws this over the fence and is to be carried over to development team.
2) IT-Architecture may not be altered or is difficult to alter.
3) Solution/software architecture role is one-to-many relationship with agile development teams.
4) IT-architecture is designed iteratively, going back and forth between IT-architects, developers and business.
5) Solution/software architecture deliverable is (nearly) finished architecture.
6) Agile team delivery is working software before a big deadline
7) Solution/software architects and developers do not or rarely interact, interaction must be organised by an external party
8) IT-architect is rarely involved in development team issues, mainly through formalised IT-architecture or quality boards
9) Solution/software architecture role is not involved in agile ceremonies
10) IT-Architecture role is/wants to be in charge of solution

[52] To present a prototype.

11) Ownership of solution transfers from IT-architecture roles to product owner/development team role
12) Development team has informal power over solution as they develop working software

In conclusion, the IT-architecture is quite rigid, but alterable with difficulties for example through a long formalised processes including high-level boards. Iterations are used to develop the IT-architecture, however these are ping-ponged over various stakeholders, following a waterfall-like or layered structure from senior stakeholders to agile developers. The balance point regarding uncertainty and risk is more on the up-front side than on the agile side, especially in the phases before actual development takes place, such as requirements and design stages. Consequently, the organisation is less mature in agile, as this phased approach to software development clashes with agile software development.

### 6.3.2. The louse in pelt model

A less IT-architecture dominant exchange model that is still on the IT-architecture side of the balance is the louse in pelt model. It can be characterised by the following characteristics:

1) IT-architecture is altered if really necessary.
2) Solution/software architecture role is one-to-multiple relationships with agile development teams.
3) Solution/software architecture deliverable an incomplete blue outline of components and their relationships.
4) Agile team delivery is working software before a big deadline.
5) Solution/software architect(s) and developers have formalised interactions through agile ceremonies and do seek out each other on their own initiative.
6) Solution/software architecture role is involved in the first iterations of the refinement process of features/stories that have not been done before, more standard cases are handled by the tech lead.
7) IT-architecture role is involved in agile ceremonies in a not-so hands-on way, for example as reviewer only.
8) Product owner is in charge of solution.

In summary, alterations are easier than in the previous model, a single solution or software architect interacts with several agile development teams through formalised interaction points in the agile ceremonies of teams, for example refinements. However, not in a hands-on way, for example as reviewer. The deliverables of solution or software architects are incomplete designs, so some risk is taken on how to approach certain components. This exchange model allows for more agile maturity than the ping-pong model as agile teams and IT-architects work together more closely. Thus, the agile processes are less conflicting with the approaches of the IT-architects, they are even integrated on a basic level. However, the focus is still more on delivering working software before a specific big deadline instead of at the end of sprint.

### 6.3.3. The solution architect as cooperating foreman

Moving from the IT-architecture dominant side to the agile dominant side, the first exchange model that comes to past is the solution architect as cooperating foreman. In this model the IT-architecture component of the combination is relatively influential as it is shared with the team lead role of the agile development team. Thus, the balance point is determined by the person fulfilling this role. This interaction model can be identified by these characteristics:

1) IT-architecture is altered regularly.
2) Solution/software architecture role is shared with team lead role.
3) Solution/software architecture deliverable an incomplete blue outline of components and their relationships.
4) Agile team delivery is shippable software at the end of a sprint.

5) Solution/software architecture role and agile developer roles have constant interaction through internalisation of both roles within the same team.
6) Solution/software architecture is involved in agile ceremonies hands-on way.
7) Product owner is in charge of solution.

Since the solution architect is the same person as the team lead, the solution architect is involved in more agile ceremonies than the previous interaction model. In addition this role is more actively involved, occasionally picking up software/solution architecture related development work for example. Shippable, working software is delivered at the end of sprints and agile maturity is relatively high. The team is self-organising as the IT-architecture knowledge is integrated within the team, the IT-architecture role can serve as an interface with other teams and introduce specific knowledge if necessary, for example from an infrastructure team. Alterations to the IT-architecture are made faster and more frequently, as the solution architect experiences them directly and communicates changes in software/solution architecture that affect other parts of the IT-architecture to relevant IT-architecture roles.

### 6.3.4. The co-development model

The most agile dominant exchange model is the co-development model. It has the following characterises:

1) IT-architecture is altered constantly
2) Solution/software architecture knowledge is integrated in the development team
3) Solution/software architecture deliverable is minimal set of decisions on components and their relationships
4) Solution/software architecture role and agile developer roles have constant interaction through internalisation of both roles within the same team
5) Agile team delivery is shippable software at the end of a sprint
6) Solution/software architecture is involved in agile ceremonies hands-on.
7) Ownership of solution is shared among stakeholders
8) Solution/software architecture is involved in agile ceremonies in a hands-on way

In this model software is software is developed by multi-disciplinary agile teams that have integrated solution, software and relevant technology architecture knowledge in multiple team members. Since the IT-architecture knowledge is shared among team-members, but not formalised in the team lead role, the interaction model has more potential to balance IT-architecture and agile software development as is needed, as the knowledge does not lie with a single person. Moreover, this single person is not the leader of the agile development team. This interaction model requires high agile and IT-architecture maturity to work properly, as team members to need be able to identify which balance point is suited based on the context. This interaction model enables high maturity of both agile and IT-architecture, as team members are hands-on involved in issues, while also communicating them effectively to other stakeholders in the organisation, being able to stand-above coordination issues themselves, or knowing who to seek out and value in this regard.

## 6.4. Classifying case studies in the new typology

In this section the classification of case studies is re-evaluated. All case studies were classified in the new typology. An overview is given is table 14.

*Table 14: Classification based on extended typology*

| Interaction model | Agile dominant | Ping-pong | Louse in pelt | Solution architect as cooperating foreman | Co-development | IT-architecture dominant |
|---|---|---|---|---|---|---|
| **Case 0** | | | x | | | |
| **Case 1** | x | | | | | |
| **Case 2** | | | | x | | |
| **Case 3** | | x | | | | |
| **Case 4** | | x | | | | |
| **Case 5** | | | x | | | x |
| **Case 6** | | | x | | | |

Case study 0 was classified as a louse in the fur interaction model. In this case study, the solution architect supported several teams, similar as in figure 10. The Scrum-master and PO were involved in the formulation of the requirements in the PSA. These requirements were translated into features, which were detailed in the refinement sessions. Workshops were organised to discuss what has been built and to discuss what should be built in the next sprints. The solution architect, PO, operations and other stakeholders, for example end-users were involved in these workshops. Prototypes were extensively used to validate requirements during these workshops. Changes to requirements that affected the PSA were iteratively added to the PEA. The solution architect was asked each week to review user stories as part of the refinement and to review deliverables during the workshops. The distance between the solution architect and agile development teams was described as small by the participant. The participant stated that this stood out from other development processes they experienced.

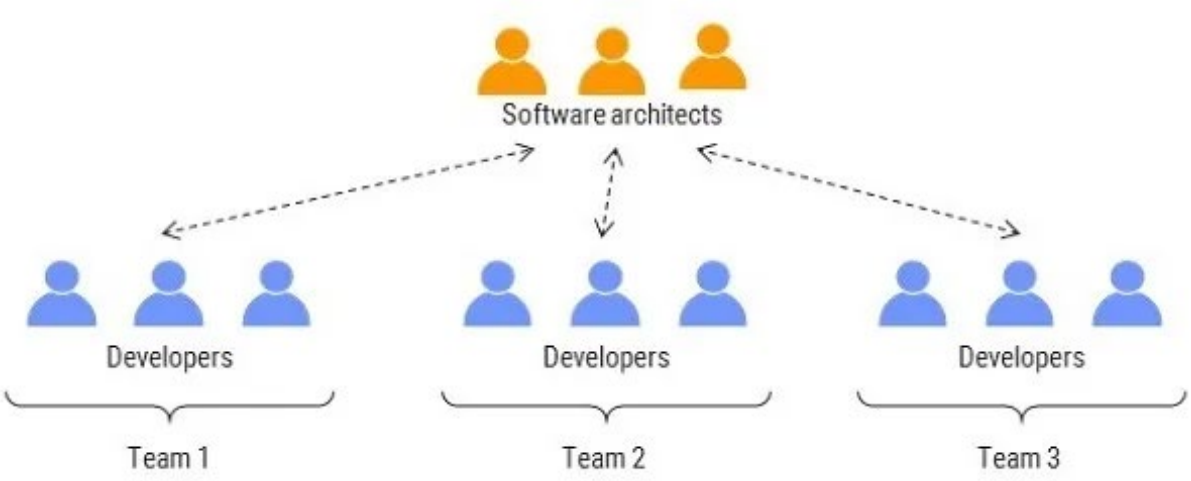


*Figure 10: Team of architecture roles supports multiple agile development teams (Mihaylov, 2015a)*

Case study studies 1 and 5 kept their classifications. Case study 1 was classified again as agile dominant. While the case study did have integrated a solution architect in without an formalised solution architect role. Up-front planning and design were not effectively used to cope with problems. For example, the team ran into issues with scalability later in the project. A more scalable setup could have prevented these issues from happening. These problems, were however more attributed to the green field, crisis situation and the PO's, than to the team. As the PO's did not act on feedback by the solution

architect and prioritised for new functionalities rather than non-functional work. Moreover, the requirements changed abnormally frequently, multiple times over a sprint. Essentially, preventing the team from achieving balance in up-front and agile architecting. An alternative explanation could be the relative inexperience of the development team in the roles of solution architect and back-end developers in combination with the crisis situation. However, the participants stated that their team was always able to deliver the expected quality on time.

Case study 2 was classified as an solution architect as cooperative foreman model. This situation is represented by figure 11. In this case study, the tech lead role was combined with the solution architect role and employed by an experienced person. This project was more stable due to the lack of a crisis structure and brown field situation. Thus, the context provided more room for the solution architect and team to effectively use up-front planning, for example by using a micro-service architecture to allow for standardisation and scalability. But the context cannot be attributed alone, as the solution architect and team initiated workshops themselves to discuss the solution architecture with all stakeholders. In addition, the team used separate refinement sessions for design and more technical stories, enabling the PO to create design oriented stories for the backlog, and the developers to transform these into technical stories at a stable rate through the sprint rhythm. Allowing the team to plan ahead 3 sprints, while keeping flexibility to alter this planning if new insights occurred, for example during the refinements of functional requirements, as here the impacts on other parts of the platform were also discussed. During the interviews, it became apparent that the team lead position granted the solution architect considerable power over the software development process, as they were the one that trained both back-end developers on the team and peer reviewed their work. All of the work at the start and selectively later, showing very hands-on involvement through the peer reviews, coding, agile ceremonies and workshops, while also being the main interaction point with other stakeholders, such as the DevOps engineer.


*Figure 11: Architecture role integrated in agile development team (Mihaylov, 2015b)*

Case study 3 was classified as a ping-pong or carry over model. The IT-architecture team was described as supporting troops in bringing alignment between the software and clients' needs by the enterprise architect. While this role also admitted that the teams might not always see the IT-architects this way. Since they set constraints or impose additional requirements, such as setting something up as a service for re-use, they were often seen as burdensome by agile development teams. Giving a clear illustration of how perspective affect the interaction model, as this diminished their recognition for the IT-architecture roles and IT-architecture. The team leads and solution architects were struggling with ownership of the solution architecture. Leading to either too little or too much solution architecture for teams. This is made worse by the lack of the software architecture layer, that specifies functional needs into technical design decisions and could address the gap between solution architecture and development teams. Moreover, participants found their organisational context, and the government context in general, not suited for agile. Since stringent deadlines, tender procedures and budgets are clashing with agile processes. Additionally, IT-architecture was described as important to lay a foundation, such as a micro-service architecture, on which the agile teams could alter the looks of the buildings in an agile fashion, hinting at carrying over IT-architecture designs from the top of the hierarchy down to the development.

Case study 4 was also identified as a ping-pong model as well, as in the first steps of the IT-architecture, IT-architects mainly interacted with business stakeholders. Then when several iterations have been done, the solution architecture was presented towards the agile teams, in a go do way. Making the business and It-architecture, callers, calling for new capabilities and functionalities and the agile software development teams producers of these capabilities and functionalities. Forgoing, the idea of

prioritising which capabilities or functionalities that are needed together. Instead the idea has been transformed into an epic hypothesis, accompanied with constraints, which were then prioritised by business. Then detailed epic designs were worked out to a solution outline, a business would be added, and an MVP would be defined. This would again be prioritised by business based on expected added value, business goals and risks, and if passed as important enough, worked out into features. Then these features would enter the backlog of the agile development teams. Resulting with an waterfall process with an agile process at the end.

Similarly, in the MVA process of case study 5, the first iteration of the MVA was with business stakeholders. The second and third-iterations were with more technical stakeholders. This would lead to a classification as carry over model as well for case study 5. The organisations architecture structure was layered according to the TOGAF model. However, the domain and solution architects did attend agile ceremonies such as refinements and demo's. This was difficult due to the fact that five teams worked in synchronised sprints. In addition, it seemed that the PSA provided the agile teams with direction for the definition of ready for their user stories. It was stated that this was an interaction point for the solution architect and agile development team, as discussions on the goal of the solutions the teams were building were held. Resulting in classification as louse in pelt interaction model. Note that case study 5 has been classified as IT-architecture dominant as well, as it involves non-agile development teams who have no assigned solution architect. Resulting in a traditional software development process for these teams.

Participants of case study 6 described that they were transitioning from waterfall towards agile software development. The degree to which this transition was implemented differed across teams. The case study was classified as a louse in pelt model as a combination of waterfall project management and agile software development existed. On the other hand, relevant stakeholders, such as PO's and solution architect were taken into a bubble that worked out designs in an agile fashion, gathering new insights along the way. However, solution architects designed the first iteration of a solution architecture and then included the tech leads, product owners and senior testers to shoot on the solution. After this the solution would help the team to create features and two or three epics. The product owner would start working out the features, epics and stories below them. The reason being, that the platform combined several technologies, which were practiced by specific teams, which provided functionalities. So an individual story did not directly add business value in this organisation. These features were more business oriented and included acceptance criteria and end-to-end outcomes. Features were reviewed by the solution architect. Product owners did not go deeper than feature level. However, this process was only used for new features and not for known cases, which were handled by the tech lead. This case study was striving for a headless, micro-service architecture to enable agile software development, especially for front-end components. Interaction between solution architects and agile development teams was mainly conducted through product owners and team leads. Solution architects were used to look 'over the wall' as illustrated in figure 10 to identify possible effects of choices by one team on another team, as teams worked independently on different technologies.

The co-creation model was not found in any of the cases, however it was formulated as the desired situation by various domain and enterprise architecture roles in case studies 3, 5 and 6. The main reason being that solution and software/solution architecture personnel is difficult to find.

## 6.5. Conclusion of chapter 6

The importance of communication and knowledge as context factors of the interaction from literature have been validated. Trust, stability and perceptions of both role were found to influence the case studies as well across several case studies. Trust, stability and perceptions of both roles were discussed to add to existing literature. The influence of uncertainty and risk as context factors to determine a balance point for agile-architecture interactions by Waterman (2018a & 2018b) has been validated. Risk and uncertainty played a role on three different aspects: 1) Requirements, 2) Technology and 3) Staffing. Context factors were able to influence the interaction of IT-architecture and agile software development in both positive and negative ways. Based on the influences of the context factors that were identified and the differences and similarities of balanced exchange interaction models, the typology has been expanded. The balanced exchange interaction model has been split up into four new exchange interaction models, resulting in a total of six possible interaction models:

1) It-architecture dominant interaction model;
2) The carry over or ping-pong model;
3) The louse in pelt model;
4) The solution architect as cooperating foreman;
5) The co-development model; and
6) Agile dominant interaction model.

Contributing with two very extreme ends and four reference points for balanced exchange models to academic literature and practitioners.

# 7. Problems in empirical interaction models

In this chapter bottlenecks as well as tensions that were identified in the case studies are discussed. Remember that a bottleneck was defined as an limitation to a desired outcome in an interaction model, while a tension concerns a discrepancy in perspective versus another part or role in the organisation. This chapter will also discuss whether they are reinforcing problems or balancing problems out.

## 7.1. Bottlenecks and tensions

Bottlenecks and tensions are discussed in relation to the interaction model of each case study.

### 7.1.1. Agile dominant

Bottlenecks and tensions for case study 1 are discussed.

**Bottlenecks**

A solution architecture role is often scarce, thus this can lead to this scarce resource being approached for issues unrelated to the role. Technical debt caused crashes, requiring hot-fixes and further increasing technical debt. Implementing these fixes put extra pressure on team-members as it imposed working over hours and increased the chance of new bugs. Over hours were also caused by a lack up-front planning caused as there was no recognition from the PO for non-functional requirements. This caused the solution architect to address critical technical debt in over hours, as these non-functional requirements were not prioritised during the planning sessions and thus could not be addressed in sprints. Frequent requirement changes during a sprint undermined both effective up-front planning and effective use of the agile methodology. This resulted in unrefined tickets entering the sprints. While adoption to changes is key in the agile methodology, changing requirements multiple times a week and assigning priority to all those changes does not allow for a development team to create a stable sprint rhythm, especially if the tickets are not well refined. Badly refined tickets were also a bottleneck for the agile teams in case of requirements that followed the normal procedure.

**Tensions**

Overhead, including solution architecture knowledge in your agile development team can cause overhead if there is too little architectural work to do. However, this was not experienced in this case study, as the solution architecture role could then pick-up development tasks. The quality was compromised by a lack of time allocated for peer reviews as well, simple procedures as this could have limited the technical debt. In case study 1, documentation was lagging behind, which caused difficulties for the development team as the solution architecture changed.

### 7.1.2. Ping-pong or carry over

Bottlenecks and tensions for case studies 3 and 4 are discussed.

**Bottlenecks**

Due to large rework phases for old projects, the possibilities for the implementation of new functionalities and development processes became limited. This bottleneck was present in both case studies. In case study 3, conflicts between certain people hampered them to do their jobs effectively, as they could not collaborate effectively, for example getting in heated discussions while they meant the same thing. The solution architect and tech lead working out technical designs together required time, however as there was a capacity issue, this tension evolved into a bottleneck in case study 3. This bottleneck limited knowledge in the organisation as well. In case study 4, business owners could not reach agreement on a of companywide aggregation and prioritisation. As everything was given management support in the first step of the process. This created a bottleneck, as there was scarcity in resources to develop the actual solution and direction level mandate was needed to override this process.

### Tensions

The software solution that was delivered in case study 3 was supposed to be an MVP, however it was not really viable, it was more a proof of concept (POC). The enterprise architect in case study saw a risk of losing oversight of the bigger picture in the nature of agile working, by dividing work into small chunks. Especially if IT-architecture were not given enough time and mandate. This occurred in case study 3, where the IT-architecture was ignored and large rework phases resulted. The argument for ignoring the IT-architecture was to save a few sprints. On the other hand, the enterprise architecture role in case study 3 admitted that there was too much room for interpretation and there were too much implicit assumptions in the IT-architecture. Making detailed designs with the tech lead would take time, on the other hand  practising ivory tower architecture also leads to endless discussions. The match of the government context and agile development were not found fit in case study 3. Fixed deadlines and large political forces increased the pressure on development teams to skim on non-functionals to save a few sprints. Stringent budgets and 1-2 year long tender procedures to acquire tooling and expertise imposed additional tensions, requiring extensive up-front planning to address associated uncertainties they imposed. Another tension in the government context for agile were the complex networks or chains of stakeholders, often with specific tasks or mandates. Similarly, governmental entities usually delegated the execution of their responsibilities to a specialised or lower entity. Which reduced the ability of the PO and agile development team to interact with end-users or the entity that initiated the development process. This tension has been found in case study 3 as well. Operations needed to know a lot of libraries in case study 3, as each team was self-organising and could choose the libraries to build new functionalities themselves. This could have been addressed by one or more software architects communicating on this aspect. It could also have been addressed by implementing DevOps, making development teams responsible for the operation of software after the development phase. This was aggravated as teams can be reallocated to another project every quarter. In case study 4, projects that required a lot of enablers were prioritised as low, even though addressing of technical debt could be included in those enablers. Political commitments, without technical consultation led to more stringent solution possibilities in this case.

#### 7.1.3. Louse in pelt

Bottlenecks and tensions for case studies 0, 5 and 6 are discussed.

### Bottlenecks

Scaling the interaction model of case study 0 was identified as difficult. As the solution architecture and agile development team roles need to be tuned into each other if they work on more than one product. Similarly, it was identified that smaller systems may seem to allow for complete up-front design by a solution architect, however issues arise if these systems need to be scaled. Additionally, the Scrum master identified that a good solution architect needs communication skills, feeling for the business, but also good technical skills to talk to the developers, which makes them difficult to find and hire. Solution architects in an agile environment need to be open to and have knowledge on one or more agile methodologies as well. As, according to this participant implementation of this interaction model would not have worked well if the IT-architects of the organisation were not open to agile. Case study 5, added to this perspective by stating that it is difficult to recruit developers for the salary and working environment that could be offered in the public sector. Similarly, an enterprise architect in case study 5 highlighted that to achieve the added value of agile, developers need to change their distribution of tasks and become more multi-disciplinary. An organisations needs to implement agile on a certain scale and for a long enough period of time to achieve the effectiveness of agile, such as predictability of burn-out charts and the learning effects for agile team members. Similar to case studies 3 and 4, in case studies 5 and 6 the impact of technical debt was not noticed by business. This is problematic as this is where the money and prioritisation came from. In case study 6, solutions architects were also a bottleneck as there was more work for them than they could absorb, reducing the self-organisation of the agile teams or even turning into an impediment if they became unavailable. This could also be an issue if a solution

architect did not manage their time well, as the diversity of their role requires them to manage their bandwidth and a risk to lose themselves into one aspect of their role. This happened in some edge cases.

### Tensions

The Scrum master in case study 0 identified an interesting tension: the agile working methodology needs to be supported by the IT-architecture. As the IT-architecture and principles should enable the agile teams to develop autonomously and independently. This tension was replicated by a participant in case study 6. As the domain lead stated that: the IT-architecture is a determining factor for the amount of flexibility that an organisation can have. While the agile approach is used to identify, prioritise, commit, implement, test and release, the actual work stays the same. For the agile process it does not matter whether the requirement has been discussed for several years or since yesterday. It needs to be refined and well understood, have acceptance criteria and solution designs to be implemented, which happens upstream. Moreover, the roles up-front determine the functional scope as well as the architectural setup: a headless, micro-service architecture enhances the flexibility. Due to this IT-architecture, complex and/or end-to-end solutions do not need to be developed for changes. While an architecture that requires end-to-end solution for new or changing capabilities strains your resources and potentially kills your flexibility, turning the supporting IT-architecture in a potential bottleneck. This is in line with the idea of Waterman (2018a) that an architecture should be able to adapt and tolerant to change.

IT-architecture roles are separated according to the TOGAF model in case study 5, however any type architect should be able to look across these boundaries, as otherwise it will be impossible to integrate these aspects. This has been identified by the participant of case study 0. This participant also identified a tendency of IT-architect too make things too standard, reducing the ability to adapt things later on. Similarly they identified that the waterfall approach could lead to a lot of rework, even though a lot of time and effort has bene put into the early stages. While the agile methodology has a risk of too much attention for the here and now and too little attention for the future.

Case study 5 illustrated this point as the domain architect stated that PSA's are either too detailed or too vague, depending on the solution architect involved, as there was no clear standard on what to put in and what not. Tensions also occurred in infrastructure enablers in this case study. Agile teams failed to address these infrastructural enablers or conditions to deliver software, such as network connections, certificates, id's etc., as they required up-front planning. There was a similar tension in resources, as new developers want to work with the tools they know, while IT-architects want a standardisation of tools. Thus this created additional difficulties in hiring and learning effects of agile teams. The tension of hard deadlines of the government context and agile methodologies has been replicated in case study 5.

The domain architect in case study 5 identified that a complete up-front architecture where everything is known did never exist. What one writes today is no longer relevant a year later. Striving for such a complete PSA is at odds with the deadlines that are present in reality. Moreover, it would end up in analysis-paralysis and one would never be able to start building, as one would always encounter new things. This participant also identified that a gap in knowledge occurred if a solution architect left. This also was the case when one leaves temporarily, for example for holidays, hampering the productivity. Since DevOps was not implemented in case study 5, documentation provided by the agile teams was often too scarce for the operations team, as explanations and images of the ideas behind the code were lacking. This was worsened by attrition. Tensions were present between IT and business, shadow-IT like robotics were the result, which imposed operational and continuity risks for the organisation. Some IT-architecture solutions were purely conceptual, while reality was different as developers needed to comply with legislation and provide business value.

In case study 6 tension between speed and stability of the solution was identified. It occurred that business stakeholders wanted the solution yesterday and did not care for stability, but IT was to blame when things broke. Additionally, the solution architecture was heavily influenced by the solution architect that designed the solution. For example by adding a fat front-end with a lot of business ruling,

adding spaghetti[53] in orchestration or hard coding to make the solution work correctly instead of designing and building a proper back-end system were ways to deliver quicker. But this created a difficult situation for maintenance if bugs did arise. The discussions with stakeholder bubbles in the iterative solution architecture approach could get very heated and comments could be taken personally as the solution designs were being challenged, however these discussions increased the solution designs quality. There was also tension in how minimal or viable an MVP needed to be, as in the past it used to be some dirty work one could throw away. But this vision shifted so that an MVP needed to be something that could be built upon. Sometimes solution architects were skipped in communication by tech leads or PO's, resulting in surprises upon end-to-end testing and in some cases knock-on problems for other teams due to dependencies the tech leads and PO's did not foresee. In case study 6 it was also identified that it was difficult to work with non-agile budget processes targets and acquisition procedures, even though it was a private sector case study. Case study 6 also experience fixed deadlines, as they operated in a regulated industry. They devised processes to deal with those issues in their interaction model and to avoid surprises. Thus, causes for mismatch in context and agile were not only present in a government context.

### 7.1.4. Solution architect as cooperative foreman

Bottlenecks and tensions for case study 2 are discussed.

#### Bottlenecks

The organisational burden was identified as high. As a lot of time was spent on planning, discussing and finding implementation for non-functional questions. Quite some time was spent on managing the process as well. Thus, while the approach was manageable for a small team, it might not scale well. The approach was demanding for stakeholders, as it required all of them to be present during the ceremonies and IT-architecture workshops. Thus, time of stakeholders could be a bottleneck for this interaction model. Since the solution architect role was with one person in this model, the solution architects needed to be available for this project whole week, as the developers were working on the project full time. Even though the solution architect worked 2,5 days on the project each week officially, because otherwise the developers could get stuck.

#### Tensions

There were architectural requirements for the platform that did not add new features. These were focused on non-functional aspects thus would show changes less fast. The Scrum meetings were of a technical nature, which could pose a challenge if the PO is not very tech-savvy. A steep learning curve was identified, as the team was self-organising. This also imposed the biggest risk: multiple team members leaving at the same time. While the distinction between design and technical stories did allow the team to work on stories from a functional perspective first and then from a technical perspective, it did create a large backlog which could fill several sprints. The team was a small island within the organisation. This posed a risk as the organisation was relatively dependent on the team. The software development process could be considered as hierarchical by developers, as the solution architect had a lot of power as the team lead. On the other hand, it did work effectively and allowed the solution architect to learn the developers defensive development processes and strategies. The short versus long term perspectives of agile and IT-architecture were replicated in this case as well. The solution architect added to this that a Scrum process makes it easier to drag each other down, than to lift each other up in terms of laziness. Due to the way the solution architect wants to see things developed, the process could be more slow and stiff. The ability to adapt imposed the risk of not meeting the planned sprint goals.

[53] Code that lacks programming style rules or has a complex and tangled control structure, resembling a bowl of spaghetti (Pizka, 2004)

### 7.1.5. Co-development

Bottlenecks for this interaction model are discussed based on the envisioned co-development interaction model by participants in case studies 3,5 and 6.

**Bottlenecks**

The team size and knowledge were identified in case study 6 as a bottleneck. With big platform solutions in an agile setup, it would not be possible to fill an agile teams with all different technologies. As this would create teams with 25-30 people, which is not suited for agile development. Additionally, finding T-shaped or Pi-shaped specialists that know all these technologies and associated processes, such as building services, was not possible. For all three case studies, the current knowledge vis-à-vis the required architecture and agile knowledge posed a bottleneck of achieving this envisioned interaction model.

### 7.1.6. IT-architecture dominant

Bottlenecks and tensions for case study 5 have already been discussed in 7.1.3. The main bottleneck for this model were issues that arose was: if mistakes were made in up-front planning and discovered later during the development process. Which led to large rework phases, additional budget and staffing required to deliver the software.

## 7.2. Reinforcing or balancing?

Tensions and bottlenecks for the carry over interaction model were found to aggravate each other, showing signs of counterproductivity. An IT-architecture that was directionally composed or inalterable led to the IT-architecture not being considered by the development team. Which in turn led to technical debt and rework, as the short-cyclical agile methodology caused the development team to lose oversight over long-term issues, for example licensing, capacity allocation, scalability, operability, security, maintainability etc. This led to quick wins in terms of development time and the opportunity to deliver functionality to the customers, but caused issues later in the process as while the POC or MVP worked, it could not be implemented as a working application (due to scaling, security. integration etc.). Thus, having no or limited business value in the long term. In these cases the agile and IT-architecture processes were affecting each other negatively instead of positively by diminishing quality. On the other hand, the louse in pelt interaction model showed that tensions or bottlenecks can alleviate each other, for example the short-term agile perspective was found repeatedly to balance out the long-term architecture perspective instead of aggravating each other. The bottlenecks for the co-development model raise the question whether this model could be implemented on a large scale at all. This question is also relevant for the solution architect as cooperative foreman model. It would be interesting to see how interaction models add value as well.

## 7.3. Conclusion of chapter 7

Tensions and bottlenecks that have been identified in case studies can shed a light on what problems were found in the interaction models that organisations used. This information adds to academic literature, as it gives substance to the typology. This information could be used by practitioners to identify the associated tensions and bottlenecks for their own interaction model.

Recurring bottlenecks across interaction models were:

- Hiring of staffing with the right knowledge.
- Scalability of agile dominant exchange models, coordination issues arose on larger scales.
- Lack of formalisation or recognition of roles.

Reoccurring tensions across interaction models were:

- Short- versus long-term perspectives in combination models.

- The IT-architecture should enable agile software development.
- Agile and the government context.

# 8. Added value in empirical interaction models

This chapter will discuss the added value first for each individual case. Then it will make cross-case comparisons. Finally it will discuss which added value had complementary interaction patterns.

## 8.1. Added value found in each interaction model.

In section the added value is discussed for each interaction model. The co-development model is shortly addressed along with the ping-pong interaction model, as the enterprise architect of case study 3 identified the added value they envisioned it would bring. Added value for IT-architecture dominant interaction model was not discussed in the interviews.

### Agile dominant

In case study 1 participants stated that they did not need to account for IT-architecture decisions as added value, however this seemed to be a sword that cut both ways, as it was also stated that the lack of priority the PO had for non-functional requirements resulted in accumulating technical debt and over hours for the development team. On the other hand, having the solution architect as part of the team helped to address more out-of-the-box issues than a team of just developers could have handled. The solution architect could join in to the conversations with the infrastructure teams as well. Moreover, the interaction model sped up development, as the solution architect developed enablers for the team that were required to create new functionalities.

### Ping-pong or carry over

Case study 3 mentioned that functional software was delivered in their case. Moreover, an improved model was proposed, where ownership transfers from the solution architect towards the lead developer as the project progresses could reduce endless discussions, which were present at the time. However, this transition of ownership was faltering in this development process. The enterprise architect did see a lot of added value in agile in terms of good communication, the refinement process, multi-disciplinary perspective on issues and division of large tasks into small pieces. Moreover, the envisioned co-development interaction model would have the added value in terms of a happier customer, more pleasant interaction between developers and IT-architects, better products and less struggle.

Not only the frequency of communication was found to be important, the means and stakeholders that were involved in communication were also important. For example, in case study 4, the solution architect interacted with business on behalf of developers on impediments. This allowed the development team to devote their time to development tasks. On the other hand direct communication might have been more effective. Another benefit was that the development team was allowed to come up with their own solutions that fit the architecture guardrails. Thus, the interaction model gave direction for developers, as requests for epics or quick wins that created more work in the long term as they did not fit the IT-architecture could be rejected by a relevant architecture board.

### Louse in pelt

Case study 0 mentioned that their louse in pelt exchange model helped the solution architect to keep close ties with reality due to the frequent interaction with the agile development teams. Additionally, the approachability of the IT-architect helped the development not to get lost in the solution architecture or their own short-cyclical way of working. The interaction model reduced strains of too much up-front design that was experienced in other, more waterfall projects by this participants. Interestingly, the participant stated that their interaction between the solution architect and agile developers helped to build things on the first try. Which conflicted with a statement by another participant in case study 3, who claimed that first time rights do not exist in IT. Technical uncertainty was approached in a stepwise manner, together with the solution architect, who played a big role in this process. Finally the solution architect could add a lot of value for the PO, as the PO had a more functional perspective, the solution architect could provide balance by representing the non-functional requirements.

In case study 5, the ability of development teams to deviate from the IT-architecture and the ability to seek how the solution could get back under the architecture again was mentioned as added value of their louse in the pelt interaction model. Note that this case study was conducted with IT-architecture roles only, which did not allow to check the agile development teams' perspective on this. However, there was varying success with this process, depending on the involved persons. Added value was also found in the ability to adjust, as there was less 'wandering in the process' than with Big Up-Front Design (BUFD), where you find out that something has been estimated incorrectly[54] when it's too late. This was mentioned as a big advantage, as they did experienced this problem with their architecture dominant development processes. Which connected nicely to the added value that roadblocks are identified early in the MVA-process. The provision of frames by IT-architecture provided added value to the agile teams as they provided the freedom that agile needs. Added value that was more to the agile side of the interaction was that an agile approach motivate people intrinsically and create learning effects, through smaller, repeated steps.

In case study 6, the cross-team stakeholder sessions to discuss the first iteration of the solution architecture was found to add a lot of value by the product manager. MVP's were also found of added value, as they could be used to determine whether a solution could actually move the needle on business Key Performance Indicators (KPI's). Then based on this outcome, resources could be allocated to this solution or not. The difference of perspective in the technology driven teams and the more broadly looking solution architect were found to be of added value as well. As the solution architect was able to look 'over the wall' for teams and identified possible knock-on effects for other teams. This way the solution architect role functioned as a bridge between teams, as they had more general knowledge of multiple different technologies, while the teams had deeper, detailed knowledge of a specific technology. The domain lead stated that the interaction model allowed the IT-architects to create a vision on what future the capabilities would have, as opposed at only formulating a vision on how to solve current product owner needs. This allowed the IT-architects to create white spots in the first place, and to fill them in the second place. The solution architect also was able to formulate a solution that avoided having to do rework. Seeking up-front alignment on requirement was found to create a deeper understanding on what was really needed, ending up with a solution that satisfied all the different needs better than a step-by-step solution would that would require a lot of modification or rework entirely. An example would be, a solution that works functionally, but kills your system in terms of performance. Something like this has been experienced in case study 3, where a system worked functionally, but was unable to cope with changes in requirements.

### Solution architect as cooperative foreman

In case study 2, added value was found in that internally everybody was up to date and there was an open and direct culture. Stakeholders could share their thoughts. This made the systems that were developed maintainable for operations for example. Since everybody was up to data, stakeholders could be transparent on issues with their superiors as well. The team could deliver added value to the PO and customers, as they were good at dealing with chance. So much that they did not require clearance (as in other cases) as they had the trust of the PO. Which allowed the team to anticipate on issues they already foresaw. This case study had almost no technical debt, and was able to use up-front planning in combination with their agile process to address future and current roadblocks.

## 8.2. Comparison across cases

A comparison can be made in the cases where the solution architect and Team lead are combined into the same role, cases 0 and 1. As one case is characterised by a lot of technical debt, while the other is not. This could be due to several things. First, the lack of technical debt could be caused by the fact that one development process was a brownfield situation, while the other case was a green field situation. Second, the experience of the Team lead/solution architect differed, as the case with technical debt had

[54] Due to uncertainty or the inability to know everything beforehand.

5-10 years of experience, while the other had 10-15 years of experience at the time of the project. Third, the stability of the environment could also be a factor, as the case with technical debt experienced a lot of new or changes in requirements during sprints due to the political nature of the project, while the other case seemed to be more independent from the politics and the requirements were more stable. Due to stability, the team was able to show the ropes to new entrants and deliver quality software. The team was also able to communicate clearly on what they would delivered, at what quality and when. This gained them trust from stakeholders like management, PO and operations. Fourth, the difference in technical debt could be attributed to the fact that in one case there was time for peer reviews, research work and work on non-functionals during sprints, while in the technical debt case there was no time for these elements. This was partly the result PO's their prioritisation, but could also be due to the ability of the team to communicate the importance of these procedures. Finally, it could be due to a combination of these things.

The MVA procedure was initiated as directionality composed IT-architectures created issues later in the development process if the wrong decisions were made at the start. The latter regularly occurred since not everything could be known beforehand, new developments occurred and things changed over time. The MVA was an iterative architecture, where in the first iteration a high-level sketch was done, for example when implementing (part of) a package, a market analysis was done to investigate how feasible the formulated requirements were, based on this the business and IT would sharpen their requirements. In the second iteration the IT-architecture team would formulate a solution architecture together with high-level design. This gave a picture of how something could be implemented and helped to formulate a plan of requirements. In the third and final iteration, the MVA turned into a PSA, since this document showed how things should be built, implemented and configured. The iterative process was found as preferable over the old situation by participants.

## 8.3. Added value with complementary interactions

Note that added value often was derived from relieving pain or problems. Participants across several cases recurred how a more iterative architecture process helped them to cope with issues of both the short-term perspective of agile development and the long-term perspective of IT-architecture. Added value was found in balancing these two approaches based on the context. The fact that no participant wanted to move to the traditional situation, could imply that the participants see complementary added value in the combination. When asked why the added value was not attributed to agile or IT-architecture alone, the domain lead of case study 6 provided a very interesting answer: *"The fact that you have an architect who doesn't talk to anybody or doesn't talk to PO's, what good does it do? Right, so as an architect, you are there to in essence figuring out how to enable a scalable, maintainable and sustainable setup that will enable us to be successful in the future as well as today. Now, how do you determine where should that future go and where you had it and etcetera? By talking to many different stakeholders and product owners are being one of the stakeholders that you normally would interact with to figure out: all right was the direction that our business is taking, right? What is it that we need to be prepared for and want to, five years in advance because the strategy is going that way. But we as an IT team don't have a capability in place in order to accommodate for that, the same time the architects play a role to a bit guide that version as well, right. So, we often encourage what we call it driven innovation. We're saying, hey, do we have some fantastic solutions that we as a team can bring as their proposal to the business to actually drive innovation and improve strategic sort of competitive edge or maybe just optimize processes and automate things, which maybe the product owners would have never thought of, because they are not aware that it exists or that sort of that's a possibility that's kind of where the architects would bring that innovative suggestion to the PO's. So, if you will say they are not talking to each other, then all of this disappears. Right. And you have two organizations that are sort of running side by side and not communicating with each other. So, it's again it's as simple as that sort of what is the power of communication."* This was the same participant that proposed the idea that the architecture setup should enable the agile teams to be flexible. This way the agile development team can create stability, by using their agile ceremonies to refine, implement and reflect on incoming work. As a back-

end developer in case study 2 stated: the agile process says nothing on the content that is developed, it creates a predictable rhythm in which work can be carried out that enables flexibility by making transparent the trade-offs that new or changing requirements bring. The other way around, the business and IT-architecture feed the backlog with requirements and user stories. These statements show how the added value of an effective agile-architecture combination could complementary.

## 8.4. Conclusion of chapter 8

It was found that every interaction model provided added value, except for the IT-architecture dominant model. For this model added value was not discussed by participants. The main complementary added value of the combination was the ability to balance up-front design with the agile process to address roadblocks or issues. This agile-architecture interactions' complementary was found in alleviating the problems that occurred when IT-architects and agile developers worked side-by-side and did not communicate effectively. Which led to either problems with sustainability of the solution, as quality attributes were not addressed or to problems with functionality, as the wrong thing had been built, since it has never been shown to an end-user.

# 9. Governance strategies and how they help to obtain complementary added value in empirical interactions

In the previous sections, descriptions of added value and problems showed that they caused balance in some cases, while added value and problems strengthened each other in other cases. Creating different types of behaviour: negative feedback loops or stabilising feedback and positive feedback loops or amplifying feedback loops. While the former offers an opportunity to provide stability but also inability to change, the latter induces change and spiralling behaviour, which can turn out positively or negatively. Thus, it was confirmed that interaction of IT-architecture and agile software development could be either complementary and counterproductive in terms of balance and cascades, depending on the implementation and context. Therefore, it is important to recognise these situations and find controls for them so that they can be used to the advantage of practitioners.

## 9.1. Coping with coordination issues in scaling

Case study 5 tried to achieve and implement an agile architecture on a larger scale through their MVA process, resulting in a louse in the pelt interaction model. Since the scale was larger in this project, as there were multiple PO's and teams involved. This introduced the problem of missing coordination over the agile teams. One participants mentioned that this sometimes caused two different teams to state the same, albeit in a different manner. Pointing out to the other they did not understand, while they meant the same thing. This seemed to be due to the 'bloodtype' that another person belonged to, for example, architects versus developers and Java developers versus COBOL developers. A domain architect then started to organise coordination across teams together with the project leader, as the previous agile implementation led to everybody being responsible for their own 'household' through their own backlog. This was difficult as resistance occurred towards IT-architects, since the development teams were in the process of building something. However, no team felt responsible to build an overarching or interconnecting services. Thus, a small management team has been put in charge to coordinate things across teams. What is interesting, is that case study 4 also used the same governance strategy to address the coordination issues that arose in making a companywide prioritisation during programme increments. In both cases, these were multi-disciplinary teams. This is something the market was already doing, for example, this can be found in SAFe. However, the government was a bit slower to adopt these things, however the need became so great that the organisation implemented this.

## 9.2. Moving away from directionally composed architecture towards iterative architecture

The MVA process[55] mentioned case study 5 is an interesting example of a governance strategy to address the issues with the directionally composed PSA procedure. Most interestingly multiple participants referred to the SAFe framework as the envisioned situation of how agile should be implemented in their organisation. This is unsurprising as adherence to this framework was a selection criterion and this framework is widely adopted in the Dutch government. There is formalisation of the framework in the PSA section of the NORA (ICTU, 2021b), to use the idea of a balance between emergent and intentional architecture when working according to an agile methodology. Through triangulation the MVA method from case study 5 and agile architecture concept from the SAFe framework have been found to be embraced by the NORA user council to *"come to diverse high-level domain architectures that are dynamic, adaptive and flexible while offering stability at the same time, without being rigid.*[56]*"* (NORA Gebruikersraad, 2022, p. 1) in a discussion on how to approach a common government domain

[55] Based on the SAFe framework on agile architecture.

[56] Original quote: *"De werkgroep GO omarmt het Agile/SAFe concept als de toekomstige manier van samenwerken. In die lijn wil de werkgroep het concept MVA (Minimum Viable Architecture) introduceren om op deze manier tot diverse domeinarchitecturen op hoofdlijnen te komen, die dynamisch, adaptief en flexibel zijn én tegelijkertijd stabiliteit bieden, zonder rigide te zijn." (NORA Gebruikersraad, 2022, p. 1)*

architecture. This formalisation of the MVA or agile architecture in the NORA can further substantiate that Dutch public organisations that were struggling with their interaction of models of IT-architecture and agile software development and were looking to move away from directionally composed architectures. Case studies 3 and 6 envisioned that not only the processes should become more agile, but that agile software development is enabled by technology as well. Technology choices such as a headless setup or a micro-architecture setup were said to enable the flexibility and self-organisation of development teams. Since, these technology choices do not require the development teams to develop complex end-to-end solutions every time, but 'pluggable' components instead. Similarly, automated tests should accompany procedures to define test and acceptance criteria. Investing on, and actively altering the technology landscape is another governance strategy that can help to move from directionally composed architecture to more self-organisation of development teams.

Likewise, case study 6 envisioned that stability of technical solutions can organised in two ways: 1) By making separate IT operations teams that ensure that the solutions stay up, running and performance according to expectations; and 2) setting up DevOps teams that are responsible for both the development and the operations cycles, aligning the incentives to build sustainable and maintainable software in these teams.

## 9.3. Addressing agile in a government context

A reoccurring theme was that agile is not suited for the Dutch government environment, as their governance structure, management and procedures do not match. This statement was widely supported among architects that were interviewed. Their reasons were strict deadlines on which laws take force, which puts very stringent deadlines on the projects. Another reason was that the management of government organisations needs to plan their budgets more than 1 year in advance, which leaves little room for agility. Other reasons were that tender procedures that caused 1-2 year delays and government budgets did not allow to hire (human) resources such as good solution architects and programmers. However, this begs the question, do these issues not also arise in a private sector environment? Do the same issues apply for those that implement legislation also apply for parties that have to comply with legislation? Is budgeting in the private sector more agile? Do large firms not work with tender-like procedures as well? To test these propositions, a private sector case study, number 6, has been added as well, to identify whether the mismatch between agile and government context exists or that there is another explanation. For example, the usual resistance that occurs when a new change is implemented. It turned out that similar problems were experienced in case study 6, as IT worked according to the agile methodology, but the rest of the organisation did not. Resulting in a hybrid situation as well. More-over this organisation also worked with a tender procedure for procurement and made budgets in advance. Another counterargument was that a fixed deadline and set of resources should not be a problem in agile development and were traditional problems of a waterfall approach. This is illustrated by Figure 8, which shows how the iron triangle of project management changes which elements are variable and fixed in an agile project (Aljaber, n.d.-a & n.d.-b). In Figure 8 resources are budget and available team members, time are deadlines for releases or milestones and scope is the work to be done, consisting of features and functionalities. In traditional waterfall development, the scope was fixed and resources and time were variable. While in agile development, the time and resources are fixed trade-offs are made in the scope. This is the fundamental added value of the agile approach, the ability to respond quickly to what is happening in the market. In other words, the product owner or manager being able to change the scope of the service based on new insights or developments while working with fixed resources towards a fixed deadline. The fact that multiple IT-architects made such claims could hint at a limited understanding of the agile methodology or an example of how both roles their perspectives clash on practical issues. Finally, governance strategies were identified to cope with mismatch in context between agile and an organisation, case study 1 for example, coped successfully by adopting a PRINCE 2 'hood' on top of their small scale agile process, as the organisation was not ready for agile at the time. While the agile methodology helped to make trade-offs on what could be developed in which timeframe transparent for stakeholders.

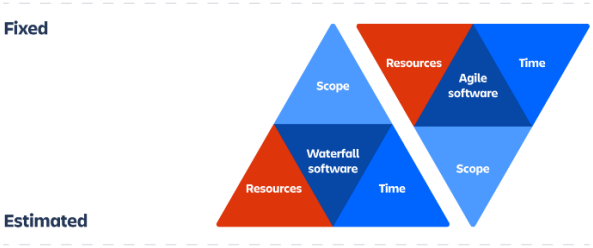


*Figure 12: Iron triangle for agile and waterfall development (Aljaber, n.d-a.)*

Strategies, proposed by a participant from the more agile mature IT section in case study 6, to cope with a non-agile context were to 1) onboard top-level leadership to agile working one of the first steps. Ensure that leaders are trained and onboarded into the agile way of working so they know what they expect; 2) Employ an agile coach if your level of agile is low, to really embed this mindset across the board. Several full-time jobs if your organisation is large; 3) Follow the process by the book. If you alter the ceremonies before you have experienced them because you don't believe in the added value, you will end up thinking and working according to your old situation, as this is comfortable for you; 4) Compliance induced fixed deadlines or quality requirements do not appear out of thin air, multiple steps and checks could be integrated in the development life cycle, including testing, quality and regulatory validation, as well as legal checks. These processes span across several roles and those roles need to recognise each other to address compliance effectively; 5) If the quality is not there for critical solutions, do not take it live into production until the quality is fixed. If it considers a small edge case, set up a multi-disciplinary team to address the issue and allocate extra budget and time accordingly.

## 9.4. Coping with a lack of resources or knowledge

Another reoccurring theme was the in-ability to hire sufficient and knowledgeable programmers and solution architects. Case studies with problems such as technical debt or large rework phases attributed these problems to a lack of interaction between solution architects and developers. This interaction was caused by understaffed solution architects. Resulting in a lack of stable access to knowledge of solution architecture for the development team. Vice versa, there was no stable access to knowledge on developer issues and planned workarounds for solution architects. This was made explicit by case study 3, as in this organisation development teams, PO's and solution architects could be reassigned to other projects on a quarterly basis. Affecting the stability of teams and changing the interaction points. These issues were aggravated by the inability to retain personal and hire new personal. These issues further complicated the relationship between the IT-architects and developers in some cases, as IT- architects felt ignored if developers just started building something that did not fit under the IT-architecture, while developers felt left in the dark by a lack of architectural guidance, but still felt the pressure of fixed deadlines. Trust then became diminished, unproductive interaction as well as monitoring and control by management started taking place, since big forces started to shift if the deadlines were not met or the quality of the solution was not sustainable upon delivery. However, these monitoring and control mechanisms were not helpful for the organisation and solution, as they left even less time and resources for productive interaction and reduced solution architects and developers willingness to take ownership of the products they delivered, pointing fingers instead. The data contained some rich descriptions of how scarcity in resources evolved into a serious bottleneck, unconstructive interaction pattern, technical debt and large rework phases as the long-term perspective was neglected or insufficiently propagated. While cases that did have access to architectural knowledge due to close contact with a solution architect or integration of a solution architect within the teams reflected more positively on interaction with architects and experience less rework and technical debt.

An interesting governance strategy to integrate architectural knowledge into the agile process through a formal process was by having an architectural review on a user story as a checklist item for the definition of ready, as in case study 0. Another example was the combination of the development

team lead role with that of the solution architect, instead of dividing these roles over separate people. In case study 3 the separation of these roles caused trouble with ownership in the translation from a PSA to a more technical solution description as this was to be done by the lead developer and a solution architect. However, since sometimes disturbed relationships were present, both the solution architect and lead developer would not take ownership (at the right time) of this technical solution description. Pointing fingers at who should do what instead. Another explanation could be that both had conflicting interests, for example the solution architect does not have to program the design and thus may care less about practical issues regarding programming or the time it takes to implement a solution. While the lead developer is not responsible for safeguarding the IT landscape and may care less about standardisation and long-term quality wins that take time away from more pressing functional requirements, especially if the long-term gains are not reaped by the lead developer. Aligning these two perspectives into one role seemed to help, as these case studies did cope with less rework and technical debt.[57]

If integration is not possible, practitioners could follow a governance strategy identified in case study 4. In this organisation an solution architect that had initially worked out an epic was tasked with safeguarding implementation of that epic. Effectively establishing a check-and-balances model where the solution architect, business stakeholders and agile development team kept each other in check. This provided clear lines of reporting for the development team and architects and aligned their incentives on the implementation of the epic. Similarly, case study 1 coped effectively with the available resources by putting the architect in a third team that supported the other two teams that worked on different parts of the same solution. This had advantages, such as increased sharing of knowledge across teams and the solution architect could do communication with external stakeholders as the solution architect had an overview of what both teams did. This strategy was replicated in case study 6, where the solution architect cut across different technology driven teams. In these case studies, the solution architects interacted with product owners and tech leads to put less pressure on the solution architecture resources.

## 9.5. Addressing the importance of formalisation and recognition of roles

Another interesting comparison could be made in the cases, 1 and 2, where the solution architect and Team lead are combined into the same role. Yet again, by whether the technical debt could be caused by the lack of formal solution architect role. Having such a formal role could help to emphasise the importance of non-functional or quality attributes of the software to the PO. In support of this theory, there is another case, number 3, in which the solution architects and developers did not always see eye to eye. This case also had difficulties in moving from to a co-development exchange model. In this case, the PO viewed IT-architecture as limiting on the ability to deliver business value and to meet hard deadlines but was also experiencing technical debt and a large rework phase. Comparing both cases, it seemed that a formally appointed and recognised solution architecture role could have helped the development team to prevent going for short-term time gains or business value delivery while neglecting the inner workings of software from a quality perspective in a broader sense than business value only.

Similarly, participants from case studies 0, 5 and 6 identified that solution architects need to recognise the practical issues that developers run into and how they should be able to think along. A formalised process, such as an architecture workshop with the team can help. Formally including the architect in the agile ceremonies seemed to help as well, as the developer in the solution architect as cooperative foreman was able to recognise the importance of the solution architecture and non-functional requirements for the sustainability and quality of their solution. Formal escalation procedures also help, but the organisation needs to ensure these have actual teeth, and are not paper tigers that result in a slap on the wrist but lack follow-up. Formalising several scenarios or types of issues and the associated levels to escalate to up-front can also help.

To enable the agile team with IT-architecture frames, the IT-architect needed to adopt an agile approach and sought dialogue with teams, stated one participant from case study 5. This architect stated

[57] Or attributed this to other reasons in case 1.

that there were enough frames to work in for developers from IT-architecture, however the compliance of the frames could be better. This was where communication and knowledge of the architect could help. In this case development teams would like to have an solution architect to visit them more often. The participant also emphasised that an IT-architect needed to seek dialogue with an MVA approach. While an enterprise architect in case study 3 stated that both the IT-architects and the development teams have an obligation to inform each other and to collect information from each other.

## 9.6. Coping with the product owner role

It was hard for the PO and development team to reach end-user or client through the layered and networked government structure. In some cases development teams and PO's did not have access to interact with the client and end-user at all, even though this is at the core of the agile methodology. Similarly, if the PO did not recognise the importance of non-functional requirements, as in case study 1, then an exchange interaction model is not viable. Thus, another likely explanation for the difference in technical debt in cases 1 and 2 is that there was no formalised architecture role that could push on these requirements and push-back if the PO wanted to go for new functionalities instead of addressing non-functional requirements. Moreover, the development team had to work overtime to address bugs and quality issues, which has likely led to even more technical debt as people got worn out. In this case the PO did also not allow to reserve time for peer reviews in the sprint, increasing the chances for bugs and technical debt. A final theory could be the inability to create a steady sprint rhythm. As the PO's in case study 1 came up with urgent and even super urgent requirements repeatedly after sprint planning and changed requirements as they saw fit on a daily basis, a steady sprint rhythm could not be achieved. Thus making it very difficult for the development team to deliver sustainable[58] functionalities. The team tried to cope by reserving capacity in sprints for urgent requests or changes and started to push back later in the process, however a formalised solution architecture role might have been able to do so earlier. In conclusion, it is very important to consider who is to be the PO and depending on the team and context. For example, whether a more technical PO is needed with understanding of non-functional requirements or a less-technical, more connected PO is necessary that can provide access to various (external) stakeholders. Shortcomings in knowledge of the PO should be compensated by team members, for example by employing a person with extensive functional knowledge to compensate for a lack of functional knowledge by a PO.

Trust was obtained by creating quality software and being transparent on when this could be expected. This seemed to go well in case studies 0 and 2, and not so well in case study 3. Trust and stability of the team(s) seemed to be linked, as case 3 both experienced instable teams as well as a lack of trust on both sides. On the side of IT-architects as low-quality software has been developed, while ignoring the It-architect(s). On the side of developers as architects were not always able to define frames well enough, defined frames felt unfeasible for the developers or do not always find the time to explain their frames/decisions. A logical explanation for the link between trust and stability is that instability could lead to lower quality software/architecture and less time to communicate with other stakeholders. Thus, reducing the trust from other stakeholders upon delivery and in the process before. It is interesting that there were mixed feelings on trust in case study 1, on the positive note is was nice to experience freedom to develop how the team saw fit, as the PO's did not care much for how their wishes were implemented. On the more negative note, the development team needed to develop parts that were not recognised by the PO's, as they did not add new functionalities directly, in over hours. Hinting a lack of trust by stakeholders. Not allocating time for non-functionals that they could not understand or see, even though the development team asked time for them. This was addressed by not accepting unrefined tickets and repeatedly addressing the importance of non-functional requirements.

[58] Scalable, maintainable, secure etc.

## 9.7. Balancing up-front and agile architecture

Case study 2 was an interesting example of how the effects from both methodologies were balanced out through implementing an agile architecture process. Case study 5 was experimenting with a similar approach on a larger scale. In case study 2, the solution architect/team lead did up-front planning to identify components and an order on how to assess them, however they usually did so together with the team and other stakeholders, such as operations, in workshops. What was interesting is that this architect left 'holes' in his blueprint that could be filled in and was open to alterations on parts of the blueprint that were not holes. This way, the stakeholders responsible for building and operating the solution could raise questions and issues, while other solutions were designed together. Sometimes a research story was time boxed and put onto the backlog, the relevant stakeholders then figured out what the options were or addressed missing information. This governance strategy was also find in case study 6. The result was that they were able to present new information or a set of options with their impact as a deliverable. This way the negative effects from upfront architecture and emergent architecture alone were balanced out, through using a balance of initial and emergent architecture. As this balance point differed for the point in time of the project, this was an agile architecture process, doing the amount of up-front and emergent development that was needed at that point in time.

## 9.8. Coping with risk and uncertainty

Multiple case studies identified that both the agile and up-front processes provide added value to address risk and uncertainty. Both processes were identified as useful in addressing uncertainty on both requirements and technology. This showed that a combination is desired to be able to respond to foreseen and unforeseen issues, as in most software development processes, both type of issues occurred. Interestingly, case study 6 used more up-front planning when establishing new functionalities they did not have experience with, a green field situation. There were also cases where they combined this with an agile approach, as the solution architecture would be made in several iterations with various stakeholders over time. Even though a participant stated that in low uncertainty and impact cases, a 100% agile approach was used to incrementally build software and solve impediments. The rationale for this was that in those cases, the organisation could take more risk. Thus, determining where to allocate the scarce solution architecture resources based on risk. Which is an interesting showcase of Waterman's (2018a & 2018b) model in practice.

## 9.9. Conclusion of chapter 9

This chapter described how governance strategies could be used to obtain complementary added value in agile-architecture software interactions by:

- Coping with coordination issues in scaling;
- Moving away from directionally composed IT-architectures towards iterative IT-architectures;
- Addressing agile in a government context;
- Coping with a lack of resources or knowledge;
- Addressing the importance of formalisation and recognition;
- Coping with the product owner role;
- Balancing up-front and agile architecture; and
- Coping with risk and uncertainty.

Multiple governance strategies have been found and linked to the added value, bottlenecks and tensions that were identified in case studies and interaction models.

# 10. Conclusion and reflection

The objective of this research was twofold: 1) to explore the relationship of IT-architects and agile software developers; 2) to develop theory on how these are influenced by governance strategies which could help practitioners to achieve outcomes they desire. The results of the previous chapters are discussed in this chapter to explain the implications of what has been found and learned. Consequently, the main results, limitations of the research, contributions to literature and practice, recommendations and opportunities for future research are discussed in this chapter.

## 10.1. Main findings

This exploratory study has identified and analysed several architecture-agile interactions in the public and private sector. Conducting this research was motivated by gaps in knowledge on if and how architecture-agile interactions were practised and affected by governance. First, it showed that based on grey and academic literature, three basic interaction models could be devised, resulting in a basic typology. Then it illustrated through multiple case studies that in practice, there was always some form of interaction between solution architecture and agile software development. The interactions of case studies were classified on the basic typology. The study then identified how the case studies were influenced by context factors found in the literature review, as well as new empirically found context factors: trust, stability and perceptions. The context factors were used to split up the balanced exchange interaction model into four new exchange interaction models, resulting in a typology of six archetypes: two extremes and four points of reference for balance between IT-architecture and agile development interactions. The case studies were reclassified using this new typology. According to this new typology problems have been identified in the form of bottlenecks and tensions. Complementarity and counterproductivity have been identified in those problems through reinforcing and balancing effects. Similarly, added value has been identified and found to be complementary. Finally governance strategies have been discussed to that help to the identified complementary added value by illustrating how they could alleviate tensions and bottlenecks.

### 10.1.1. Generalisability of results

This study hints that the need for agile-architecture interactions are present in a broader context than the public sector alone, as a the private, corporate sector case study identified similar problems. However, this needs to be validated through a study that investigates various organisational types in the private sector and in mixed forms of public private partnerships, as well as varying sizes of organisations.

To cope with the small sample size and issues this can create, multiple interviews have been conducted for each case study and multiple case studies have been performed. It is also worth noting that not all researchers agree on the limited ability of even a single case study, as even extreme cases can be useful to make issues transparent that might also be at play in less extreme cases for example (Flyvbjerg, 2006). Thus this single case can then be used as a reference for other, but similar cases. Thus, contributing to science.

The interviews showed that practice was both different and similar to theory. Falessi et al., (2010) identified that theoretical frameworks, ideas or principles to define roles, responsibilities and rights were used as reference points, they were often not achievable in practice due to practical matters of adoption. This finding has been replicated. In practice, practitioners tried to cope with these limitations, in ways that were not discussed in theory. Effectively tailoring theory to their own situations, as in most case studies, respondents answered that they had implemented a framework with their own 'sauce' to tailor the framework to the organisational or development processes' needs. This study has also replicated findings on context factors of studies by Yang et al. (2016) and of agile architecting by Waterman (2018a & 2018b). The interviews also showed that the combination or the interaction of IT-architecture and agile software development ranged from a theoretical concept to something so obvious it seems a bit strange to inquire on in an interview. This variety of implementations and responses contributed to the generalisation of new findings, while replication of various interaction models,

problems, added value and governance strategies indicate that these findings are generalisable beyond the case studies.

## 10.2. Limitations

This section will discuss limitations of the findings. This section will also be used to evaluate the theory building process and discuss limitations of my own theory building process (as opposed to general limitations discussed in Chapter 3).

### 10.2.1. Multi-case study approach

There were practical limitations: the availability of participants versus the research timeline, assessing the expertise of the participants, assessing the tone[59] of the participants, inquiring on topics[60] and the time that interviewing and transcribing took. There were also methodological limitations, such as the small sample size which could be said to reduce generalisability and the difficulty to analyse the amount and richness of interview data. Finally, identification of problems was difficult in some cases due to the preference of participants to give socially acceptable answers.

Since the interviewees were inquired on working practices and problems in their organisations and with their colleagues, they have been promised anonymity of their organisations and themselves. The transcripts have been typed by hand by the researcher, which took up a considerate amount of time that could have been spent on more thorough analysis, data collection or reporting. Since availability of interviewees was a limitation, a trade-off has been made between comparative capacity of cases and availability of interviewees with respect to the timeline of the research. While the focus was on software architects, solution architects, enterprise architects and domain architects were interviewed instead in several cases. As not every organisation had a solution or software architect. But also since these roles were understaffed. However, due to this decision more opportunities arose than time allowed to investigate and the opportunity arose to select cases which were most likely to provide new or rival insights, such as the private sector case. Moreover, flexibility to ask different questions to various roles was a great benefit, as some roles could shed light on the up-front design processes, while others could explain more on the development processes. Similarly, slight differences in case study selection as well as discussions with supervisors helped to identify rival hypotheses and reduce researcher bias. Thus, the benefits of the multi-case study theory building methodology have been successfully exploited to address issues associated with small sample size. To the inclusion of multiple case studies, the effects of limitations such as assessing the expertise and tone of participants became less influential, as contradictions made themselves prevalent across cases.

A major limitation of this study was that participants had difficulty in defining interactions between IT-architecture and agile software development. As mentioned, the cases differed greatly in richness of description of the interaction models. Some interviews were too short to discuss this interaction, it's added value, problems and its relationship with governance, while others the interviewees quickly glossed over the interaction models and its effects, unopen to further inquiry and probing by the interviewer. On the one hand, this makes the case studies difficult to compare, on the other hand the (in)ability to give further details is telling as well.

The vastness and richness of data were a limitations and a blessings in this study. Especially in respect to the timeframe of 20-25 weeks and the sensitivity of data which did not allow external tooling to lower the burden of transcribing and analysing the interviews. Data in the form of over 18 hours of recordings and 180 pages have been collected, which made transcription and analysis long and difficult processes. The case study logbook, colour coding of transcripts, discussions with the research team and

[59] Some people tend to be overly pessimistic or make statements on problems, while other tend to be overly positive, making things seem better than they are.

[60] This concerned both working practices, which is information that organisations do not want to be public as well as inquiring participants on problems within their organisations and elicitation of critical judgment of their colleagues. These types of questions tend to invoke socially acceptable answers, especially if the transcripts were to be made publicly available.

summarising insights into tabular displays helped to cope with the vastness and richness of the data but were time-consuming processes, as determining what could actually be an insight required shifting back and forward through this data on seven different organisations and over twelve roles. Similarly, each new case study could add new insights, which could turn previously irrelevant statements to relevant ones. This led to rereading, analysing and summarising the transcripts multiple times. On the other hand, a lack of rich data or interview participants were serious threats for this study, as this would have reduced the validity of the individual cases, as well as the ability to generalise and replicate theory across cases. So, while the number of cases was a benefit on one hand, it had the implication that each case received less attention than if a lower number of cases would have been used. Moreover, cases tended to focus on either the high level architecture facets from enterprise architecture and senior stakeholders or on very operational facets of solution architects and developers. Therefore, a lack of triangulation on the interaction of this higher, organisational level with the operational level is present. On the other hand, many of these senior stakeholders did not interact with developers at all and vice versa. Also, the operational and more senior levels could be compared across cases.

This research has been carried out by a researcher that followed an educational internship at a company. This company internship has biased the case study selection. However, the company also provided access to data which the researcher did not have. To address the potential of too much similarity in selection, participants have been approached through a variety of people working at different departments in the company.

### 10.2.2. Influence of perspectives

In this specific research, which role said what is important to identify whether this is a perspective or a fact. As even the questions which seemed to be factual were prone to perspective or framing. For example, when asking whether the project was delivered within time and budget, I often got the answer: it depends on how you look at it, we had a working product, but were prolonged to improve/repair the product. I would take this indistinguishability further by stating that there are facts within the data, but it is impossible to distinguish them from perspectives due to the nature of the research. A logical replication of a (reoccurring) perspective is the closest one can come to a fact in this research and these are biased by the perspective of the interviewee as well as of the interpreter.[61] Nevertheless, this research is a contribution to science as through performativity perspectives can affect reality. For those that do not believe in this phenomenon, the identification and publication of these perspectives can serve as starting points for identification of facts.

## 10.3. Research contributions

This section discusses the contributions to scientific literature and as contributions for practice.

### 10.3.1. Scientific implications

This thesis used multiple case studies to update knowledge of interactions between IT-architects, agile developers and governance. While this is partly replication of earlier theory, such as influence of contextual factors identified by Yang et al. (2016) and Waterman's (2018a & 2018b) diptych, there have been made various new contributions which were replicable across cases. New context factors: trust, stability and perceptions as well as the typology, governance strategies and complementary nature of added value and problems. The diptych by Waterman (2018a & 2018b) has been validated by showing how the agile architecture is put into practice. Showing which roles are included at which stages of designing an agile-architecture in various organisations, which were identified in the literature review as a gap in knowledge. Falessi et al. (2010) stated that issues of adopting agile-architecture interactions resided in practical matters of adoption. The complementary nature of added value and problems in the interaction, the typology and governance strategies are new contributions to academic literature in this regard. The typology could be further developed or expanded by other researchers. The relationship between governance and interaction models was not as simple as the make or break relationship

[61] Researcher, but also reader.

presented in the beginning of the results section. For example, in some cases formal governance procedures could be circumvented as power resided with those who delivered working software. While in other cases the governance strategies led to complementary added value. Practical detail has been given to the theoretical notion of interaction through the typology and accompanying characteristics. This thesis has updated the scientific knowledge repository by presenting new archetypes that can be used for case-based reasoning with practitioners in other, but similar situations. Flyvbjerg (2006) stated that expert knowledge is achieved by having internalised many distinct cases through experience and literature. This way the study has improved on grey literature in the classification of interaction models and added empirical evidence for complementary relationships between architecture and agile to scientific literature. As opposed to architecture or agile frameworks alone and the high-level interaction model of the SAFe framework, this study has split this balanced exchange model into four archetypes based on the empirical case studies. Of which one, the co-development model, needs yet to be identified in practice. Similarly, it would be interesting to investigate whether the archetypes can be found in mixed public-private, medium and smaller private organisations, like scale- and start-ups.

### 10.3.2. Practical implications

This study provides an opportunity for experts to internalise new knowledge through the seven new case studies. In order to gain interview time from their scarce resources, organisations that were experiencing problems due to their interaction models asked to share and present the findings of this research to learn from other organisations. Illustrating how knowledge of other case studies could help practitioners to address their problems and provide practitioners with the ability to pursue or avoid interaction models through governance strategies. The case studies clearly illustrated that what works for an organisation is context dependent and gives practical examples of how each case had its own problems and added value, how added value was situational and optimisation existed only locally. If one is to look at the effects on the organisation, operation, client, end-user and wider context, only trade-offs exist. Therefore, it is up to the practitioner to make an informed decision on which interaction model and associated added value and problems they wish to pursue. This research has made the trade-offs more transparent for practitioners. Chapter 6 provides practitioners with the opportunity to investigate which archetype is closest to their own organisation. Chapters 7 and 8 allow them to identify which problems and added value others experienced. Chapter 9 provides practitioners with several governance strategies for various distinct use cases and with governance strategies are applicable in multiple interaction models. Chapters 6-9 could also be used to identify which archetypes is more desirable than a practitioners current situation.

Participants from multiple case studies stated that integration of architectural knowledge into development teams was something they desired but were not able to accomplish due to scarcity in resources. Based on this it is recommended to practitioners to formally integrate an solution and software architecture role into each development team if resources allow so. If resources do not allow so, smart choices could be made, looking at which teams could share a supporting team of experts. Looking for teams that need to integrate their services is helpful in this case, as the solution and software architecture role can keep an overview of integration of the teams their work and communicate with external stakeholders. It is advised to keep the number of teams a solution or software architect supports as low as possible and to take other measures that allow attendance of these architects to agile developer rituals, such as organising workshops with multiple teams instead of working in isolated, synchronised sprints.[62] Integration of the solution and software architecture role could also be done by assigning co-ownership of these architectures and software deliverables or by formalising interaction through procedures or meetings to kick-start the interaction process between both roles. For example by adding an architectural review as a requirement for the DoR of a user story. On the other hand, the study showed that separation of solution architecture and development roles could help to keep both roles in check through a checks-and-balances system and comes with the potential to look over the wall. This, has potential to affect the

[62] These make it practically difficult for an architect to attend, as synchronisation of isolated sprints would mean that the architect has to follow multiple meetings on the same day or even at the same time.

perspective of practitioners as well. Seeing how conflict, discussion or extra work adds value for the business or quality of software.

This study has potential to shift the perspective of practitioners that agile software development and IT-architecture are not suitable for combination in a public context. Combining the fact that similar problems and added value were found in the private sector with the fact that many IT-architects made this claim as well as the falsified claim that agile does not match well with fixed deadlines, I would recommend to invest in training for architects in working in an agile environment to show that balance can be obtained in stability and flexibility. Similarly I would recommend to train agile developers in IT-architecture knowledge and the benefits that are working together or under an IT-architect(ure) can bring in terms of viability and sustainability of the software that is created. Training both perspectives makes interaction with the other role more accessible, as one knows better which issues could be interacted on. In addition, training could help to identify which technological changes are needed to accommodate agile working practices and self-organising teams. Moreover, the change in practitioner perspective is the first step towards better quality software, developed in a reduced timespan and for a lower budget. While the government should invest on quality education in both deep and broad knowledge of IT. Employers should also see training or education of their personnel in technical as well as methodological matters of IT as an investment. Similarly, the government can increase its agility by providing training in both the technical and methodological best practices, to integrate knowledge on how agility can be achieved in the organisation. Touch points to increase agile maturity in the public sector would be technical architectures and architecture processes that enable agility, onboarding of top-level management of government organisations to the agile working methodology and new organisational structures such as DevOps teams.

Finally, the study shows that it is important to create trust between both roles to ensure cooperation. I would recommend practitioners to be sensitive of actions that built or decrease trust from other stakeholders. For example, ignoring the architecture for a short-term time or resource gain can decrease trust of architects in the development teams. Similarly, stating that the way something has been built in an unconstructive manner, without taking the time to think alongside with the team to what could be feasible or taking a long time to deliver a fully fleshed out architecture instead of some key decisions on which the team can act can decrease the trust of agile developers in IT-architects. It is important for practitioners to realise that other stakeholders are not there to make your life difficult, but strive to create quality software from their perspective as well. One developer stated that trust could be built by being transparent on what you will deliver, when you will deliver, at which quality and to make this a predictable pattern by delivering quality consistently. An IT-architect stated something similar, by stating that it is sign of weakness to reject an idea of a development team without a conversation on why they think this is the right way, how this would affect the organisation and being open to the perspective of the developer on architectural decisions. Teams that reflect positively on their development process described that they had trust and mandate from stakeholders to do what they thought was necessary, while teams that reflected more negatively on their development found this lacking and experienced more technical debt and rework later.

## 10.4. Recommendations for future research

The hypothesis that exchange models can create an upward spiral in terms of trust and the ability to deliver quality software could be tested in an experimental sense, as there has been no investigation on which software development process delivered most quality in respect to the resources and time used, as this was not the point of the research. Comparing these cases on qualities comes with its own limitations, for example being heavily dependent on the definition of quality. An in-depth single case study of one organisation with direct observation could provide further evidence for or against the theory that complementary added value could be obtained from architecture-agile interactions in software development. A multi-case study could also be used but will require more resources, such as a research team to facilitate discussions and cope with the amount of data within a reasonable timeframe.

Case studies that take into consideration more roles and perspectives from the same organisation

would add to validity. For example, the perspective of external parties such as end-users and clients missed in this study. Thus, a more detailed study could interview a client, PO, program manager, enterprise architect, solution architect, involved development teams, end-users and various business stakeholders to capture the full journey from a business idea or legal text to a working software product. This would require a researcher that is close to, but independent of, the organisation to gain access while avoiding bias. Similar case study research is needed to identify whether the interaction models defined in the typology can be replicated.

Problems were encountered when working according to agile methodologies, however, are the problems caused by agile or it's interaction with architecture? Or is this methodology making problems that were previously also involved in software development more observable? Agile methodologies were developed to close the gap between developers, the end-user(s) and client. However, this does not solve further problems in software development, such as the transition from development to operations, or influences from the 'outside world' that require a certain level of quality for a certain budget and delivered in a certain time. Nonetheless, the majority of organisations in the case studies were working according to an agile methodology and successfully implemented basic agile processes such as a backlog, daily stand-ups, refinement sessions and sprints, however stated that further improvements could be made in their software development processes, such as implementing DevOps[63] or creating more interaction on the formulation of the solution architecture. The difference in agile approaches and maturity turned out to be more of an opportunity than a threat, as it led to the identification problems as well as added value in transitions from waterfall development towards agile. Making apparent the interaction between traditional and agile favoured IT-architects.

While, most modern agile methodologies propagate the creation of an agile architecture[64] and multi-disciplinary development teams that include all stakeholders that can affect or will be affected by the solution: IT-architects, developers, testers, business stakeholders, end-users, etc. The fact that these best practices were not implemented begs the question what this research would have found if these best practices were implemented. Therefore, a study with a longer time span, more researchers in order to cover a broader range of organisations or a case study approach focused on an exquisite development organisation(s) could yield additional insights on how affected this study is by the selection of cases. This study could then have contributed by exploring facets which could determine what an exquisite development organisation would look like and how it would (not) approach software development through the exploration of interaction models, outcomes of development processes, governance strategies and the added value and problems that they created.

From a governance perspective it would be interesting to investigate the role of the NORA user community and whether the adoption of the agile architecture process is top-down, bottom-up or a combination of the two. Moreover, it would be interesting to investigate the effect of having a user community that administers a national government reference architecture on software development and compare the Dutch approach with other countries.

[63] In DevOps, the developers are responsible for keeping the system operational after development. This reduces the gap between developers and operations and should create software that is easier to maintain, monitor, alter etc.

[64] An architecture that allows for options, as not everything can be known beforehand, and even if, then there will be changes in the outside world that affect the architecture over the course of the development process.

# Literature


Aljaber, T. (n.d.-a). *Agile Iron Triangle* [Illustration]. Atlassian. https://www.atlassian.com/agile/agile-at-scale/agile-iron-triangle

Aljaber, T. (n.d.-b). *The iron triangle of planning*. Atlassian. Retrieved June 2, 2022, from https://www.atlassian.com/agile/agile-at-scale/agile-iron-triangle

Alsahli, A., Khan, H., & Alyahya, S. (2016). Toward an Agile Approach to Managing the Effect of Requirements on Software Architecture during Global Software Development [Article]. *Scientific Programming*, *2016*, 16, Article 8198039. https://doi.org/10.1155/2016/8198039

ANP, & Doorenbosch, T. (2019, December 4). *Software-update legt informatie NS plat*. AG Connect. Retrieved June 9, 2022, from https://www.agconnect.nl/artikel/software-update-legt-informatie-ns-plat

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., & Jeffries, R. (2001a). *Manifesto for Agile Software Development*. Agilemanifesto. Retrieved June 28, 2022, from https://agilemanifesto.org/

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., & Jeffries, R. (2001b). *Principles behind the Agile Manifesto*. Agilemanifesto. Retrieved June 28, 2022, from https://agilemanifesto.org/principles.html

Bellomo, S., Gorton, I., & Kazman, R. (2015). Toward Agile Architecture Insights from 15 Years of ATAM Data [Article]. *Ieee Software*, *32*(5), 38-45. https://doi.org/10.1109/ms.2015.35

Bloch, M., Blumberg, S., & Laartz, J. (2012, 12-01). Delivering large-scale IT projects on time, on budget, and on value. https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value

Scaled Agile, Inc.. (2021, April 27). *SAFe Glossary*. Scaled Agile Framework. https://www.scaledagileframework.com/glossary/

Bourgeois III, L. J., & Eisenhardt, K. M. (1988). Strategic decision processes in high velocity environments: Four cases in the microcomputer industry. *Management science*, *34*(7), 816-835.

Breivold, H. P., Sundmark, D., Wallin, P., & Larsson, S. (2010, 2010). What Does Research Say about Agile and Architecture?

Burgelman, R. A. (1983). A Process Model of Internal Corporate Venturing in the Diversified Major Firm. *Administrative Science Quarterly*, *28*(2), 223–244. https://doi.org/10.2307/2392619

Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A framework for adapting agile development methodologies. *European Journal of Information Systems*, *18*(4), 332-343. https://doi.org/10.1057/ejis.2009.26

Chulani, S., Williams, C., & Yaeli, A. (2008). Software development governance and its concerns Proceedings of the 1st international workshop on Software development governance, Leipzig, Germany. https://doi.org/10.1145/1370720.1370723

Cleland-Huang, J., Hanmer, R. S., Supakkul, S., & Mirakhorli, M. (2013). The twin peaks of requirements and architecture. *Ieee Software*, *30*(2), 24-29.

Creswell. (2009). The Selection of a Research Design. In *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (pp. 3-18).

Curasi, C. F. (2001). A Critical Exploration of Face-to Face Interviewing vs. Computer-Mediated Interviewing. *International Journal of Market Research*, *43*(4), 1-13. https://doi.org/10.1177/147078530104300402

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of management review*, *14*(4), 532-550.

Falessi, D., Cantone, G., Sarcia, S. A., Calavaro, G., Subiaco, P., & D'Amore, C. (2010). Peaceful Coexistence: Agile Developer Perspectives on Software Architecture [Article]. *Ieee Software*, *27*(2), 23-25. https://doi.org/10.1109/ms.2010.49

Flyvbjerg, B. (2006). Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*, *12*(2), 219–245. https://doi.org/10.1177/1077800405284363

Fukuyama, F. (2016). Governance: What do we know, and how do we know it? *Annual Review of Political Science*, *19*, 89-105.

Glaser, B. G., & Strauss, A. L. (1967). The Discovery of Grounded Theory-Strategies for Qualitative Research (London, Weiderfeld and Nicolson). *Přejít k původnímu zdroji*.
Gong, Y. (2012). Engineering flexible and agile services: a reference architecture for administrative processes.
Gong, Y., & Janssen, M. (2019). The value of and myths about enterprise architecture. *International Journal of Information Management*, *46*, 1-9. https://doi.org/10.1016/j.ijinfomgt.2018.11.006
Gong, Y., & Janssen, M. (2020). Exploring Causal Factors Influencing Enterprise Architecture Failure. In (pp. 341-352). Springer International Publishing. https://doi.org/10.1007/978-3-030-64849-7_31
Hanschke, S., Ernsting, J., & Kuchen, H. (2015). Integrating Agile Software Development and Enterprise Architecture Management. *2015 48th Hawaii International Conference on System Sciences*. https://doi.org/10.1109/hicss.2015.492
Harris, R. (2021, May 20). *Agile vs. Waterfall* [Illustration]. LinkedIn. https://www.linkedin.com/pulse/software-development-life-cycle-sdlc-tutorial-richard-harris/
ICTU. (2021a, October). *NORA Online*. noraonline. Retrieved February 16, 2022, from https://www.noraonline.nl/wiki/NORA_online
ICTU. (2021b, December 29). *NORA (Nederlandse Overheid Referentie Architectuur)*. DigitaleOverheid.nl. Retrieved May 30, 2022, from https://www.digitaleoverheid.nl/overzicht-van-alle-onderwerpen/standaardisatie-en-architectuur/nora/
ICTU. (2022, January 13). *PSA (Project Startarchitectuur) - NORA Online*. noraonline.nl. Retrieved May 30, 2022, from https://www.noraonline.nl/wiki/PSA_(Project_Startarchitectuur)
Irvine, A. (2011). Duration, Dominance and Depth in Telephone and Face-to-Face Interviews: A Comparative Exploration. *International Journal of Qualitative Methods*, *10*(3), 202-220. https://doi.org/10.1177/160940691101000302
ISO. (2011). ISO/IEC/IEEE Std 42010-2011. In *Systems and software engineering – Architecture description*.
Kable, A. K., Pich, J., & Maslin-Prothero, S. E. (2012). A structured approach to documenting a search strategy for publication: A 12 step guideline for authors. *Nurse education today*, *32*(8), 878-886.
Lankhorst, M. (2009). *Enterprise architecture at work* (Vol. 352). Springer.
Lankhorst, M. (2019, January 9). *Four-layer incarnation of SAFe* [Illustration]. Bizzdesign. https://bizzdesign.com/blog/agile-architecture-using-archimate-with-the-scaled-agile-framework-safe/
Madison, J. (2010). Agile-Architecture Interactions [Article]. *Ieee Software*, *27*(2), 41-48. https://doi.org/10.1109/ms.2010.35
Mihaylov, B. (2015a, December 28). *A Separate Team of Software Architects Works with Multiple Development Teams* [Illustration]. InfoQ. https://www.infoq.com/articles/towards-agile-software-architecture/
Mihaylov, B. (2015b, December 28). *Each Development Team Has One Software Architect* [Illustration]. InfoQ. https://www.infoq.com/articles/towards-agile-software-architecture/
Mihaylov, B. (2015c, December 28). *Towards an Agile Software Architecture*. InfoQ. Retrieved June 18, 2022, from https://www.infoq.com/articles/towards-agile-software-architecture/
Mihaylov, B. (2015d, December 28). *Traditional waterfall model* [Illustration]. Infoq. https://www.infoq.com/articles/towards-agile-software-architecture/
Miles, M. B., & Huberman, A. M. (1984). Qualitative data analysis. Beverly Hills.
Mintzberg, H. (1979). An Emerging Strategy of "Direct" Research. *Administrative Science Quarterly*, *24*(4), 582–589. https://doi.org/10.2307/2392364
NORA. (2019, October 8). *NORA bindende afspraken*. Digitale Overheid. Retrieved June 26, 2022, from https://www.digitaleoverheid.nl/achtergrondartikelen/nora-bindende-afspraken/
NORA Gebruikersraad. (2022, March). *NORA Gebruikersraad/2019-03-26* (No. 2019–03–26). ICTU. https://www.noraonline.nl/wiki/NORA_Gebruikersraad/2019-03-26
Parlementaire ondervragingscommissie Kinderopvangtoeslag. (2020, December). *Ongekend Onrecht* (No. 35510, nr. 3). https://www.tweedekamer.nl/sites/default/files/atoms/files/20201217_eindverslag_parlementaire_ondervragingscommissie_kinderopvangtoeslag.pdf

Pfeffer, J. (1982). *Organizations and organization theory* (pp. 237-251). Boston: Pitman.
Pizka, M. (2004, June). Straightening spaghetti-code with refactoring?. In *Software Engineering Research and Practice* (pp. 846-852).
Poort, E. R. (2014). THE PRAGMATIC ARCHITECT Driving Agile Architecting with Cost and Risk [Article]. *Ieee Software*, *31*(5), 20-23. https://doi.org/10.1109/ms.2014.111
Smite, D., Van Solingen, R., & Chatzipetrou, P. (2020). The Offshoring Elephant in the Room: Turnover. *Ieee Software*, *37*(3), 54-62. https://doi.org/10.1109/ms.2018.2886179
The Open Group. (n.d.). *ArchiMate® 3.1 Specification*. Retrieved February 16, 2022, from https://pubs.opengroup.org/architecture/archimate3-doc/toc.html
The Open Group. (2018, April). *The TOGAF Standard, Version 9.2 - Content Metamodel*. Retrieved February 16, 2022, from https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap30.html
Waterman, M. (2018a). Agility, Risk, and Uncertainty, Part 1: Designing an Agile Architecture [Article]. *Ieee Software*, *35*(2), 99-101. https://doi.org/10.1109/MS.2018.1661335
Waterman, M. (2018b). Agility, Risk, and Uncertainty, Part 2 How Risk Impacts Agile Architecture [Article]. *Ieee Software*, *35*(3), 18-19. https://doi.org/10.1109/MS.2018.2141017
Woods, E. (2015). Aligning Architecture Work with Agile Teams [Article]. *Ieee Software*, *32*(5), 24-26. https://doi.org/10.1109/ms.2015.119
Yang, C., Liang, P., & Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development [Article]. *Journal of Systems and Software*, *111*, 157-184. https://doi.org/10.1016/j.jss.2015.09.028
Yin. (n.d.). A (very) brief refresher on the case study method. In *Case Study Research* (pp. 3-20).
Yin, R. K. (1981). The case study crisis: Some answers. *Administrative science quarterly*, *26*(1), 58-65.
Yin, R. (1984). case study research. Beverly Hills.
Yin, R. K. (2018). *Case Study Research and Applications: Design and Methods*.

**Personal communication with supervisors:**

Consultation with supervisor M.F.W.H.A. Janssen. (2021). Full Professor in ICT & Governance at TU Delft.
Consultation with supervisor W.G.P. Heijnen. (2022). Senior Manager Digital Transformation public sector at KPMG.
Consultation with supervisor H.G. van der Voort (2022). Assistant professor in Organisation & Governance at TU Delft.

# Glossary

**Administrative/management organisation:** party to which ownership of the software solution is transferred after development.

**Business landscape:** Collection of documents, designs and visions that describe the current and future business processes and capabilities of an organisation.

**Client:** party that the development team is building a solution for.

**Customer:** person that will have to work with the software solution, end-user.

**Governance strategy:** Formal policy that assigns roles and responsibilities or monitors or enforces these roles and responsibilities. Management decisions, governance structures, frameworks, rituals and procedures are manifestations of governance strategies.

**IT-architect:** Since architect is a protected term related to the construction of buildings, a preposition had to be used for the term architect. Even though in literature and practice the term architect is often used to discuss the collection of various architecture roles: enterprise, business, technology, solution, software, infrastructure etc., in this research the word IT-architect is used for this. This does have the unfortunate implicit assumption that only the IT-side of the architecture roles are considered and not the business architecture role(s), however these are included in the IT-architect term, for lack of a better term.

**IT-architecture:** Similarly to IT-architect, the word IT-architecture is used to discuss the products of several architecture roles in an organisation, for example the enterprise architecture is the product of an enterprise architect and an IT-architecture is a collection of an enterprise, domain and solution architecture. Thus, an IT-architecture are the products of various architecture roles, with the same unfortunate implicit assumption that the business architecture is not included in this collection, while it is.

**IT-architecture landscape:** Collection of IT-architecture documents, designs and visions that describe the current and future IT capabilities of an organisation.

**Operations:** party to which ownership of the software solution is transferred after development.

**Product owner:** Stakeholder, usually from the business side of the organisation that takes responsibility for the solution and determines together with the development team which user stories are worked on in which sprint.

**SAFe:** Scaled agile framework, a framework designed to help *organisations* cope with issues that arise when agile is implemented with multiple teams that work on the same solution. Builds upon Scrum and is tailored towards the whole organisation.

**Scrum:** Agile framework for *small teams*.

# Appendix A – Quality assessment of papers

Table 15 gives an overview of all the papers that were assessed by hand with the inclusion and exclusion criteria.

*Table 15: Overview of papers selected for quality assessment*

| ID. | Author, year | Selection method | Quality appraisal: include/exclude |
|---|---|---|---|
| **1.** | Gong & Janssen, 2019 | Manual/ supervisor | Exclude, already used for introduction. |
| **2.** | Gong & Janssen, 2020 | Manual/ supervisor | Exclude, already used for introduction. |
| **3.** | Luna et al., 2019 | Manual/ supervisor | Exclude, focused on agile governance theory, enough papers about the interaction of agility and architecture are available |
| **4.** | Cao et al., 2009 | Manual | Exclude, pre 2010 |
| **5.** | Ashrafi et al., 2019 | Search query 2 | Exclude, only focused on how business analytics can increase agility. |
| **6.** | Larson & Chang, 2016 | Search query 2 | Exclude, paper is a review of how agile practices have evolved with business intelligence. |
| **7.** | Liu et al., 2013 | Search query 2 | Exclude, not about software architecture and agile software development |
| **8.** | Dingsoyr et al., 2012 | Search query 2 | Exclude, Falessi et al., 2010 is referenced to and already discusses these issues. |
| **9.** | Spijkerman, 2021 | Search query 3 | Exclude, paper proses a framework to align requirements engineering |
| **10.** | Kisimov et al., 2020 | Search query 3 | Exclude, specifically focused agile architecture for big data |
| **11.** | Waterman, 2018 part 1 | Search query 3 | Include, paper is about a dilemma software architects face in agile environments. |
| **12.** | Waterman, 2018 part 2 | Search query 3 | Include, discusses the risks associated with the outcome of part 1. |
| **13.** | Angelov & Beer, 2017 | Search query 3 | Exclude, focused on software architecting in agile projects in education. |
| **14.** | Aslahli et al., 2016 | Search query 3 | Include, discusses an integration of agile practices in architecting practice for software development. Does use the Twin Peaks model, however as a base to design a new approach to simultaneously handle requirement and architecture changes. |

| | | | |
|---|---|---|---|
| **15.** | Yang et al, 2016 | Search query 3 | Include, systematic mapping study of literature about architecture-agility combination. |
| **16.** | Bellomo, 2015 | Search query 3 | Include, review of agile community refocusing on approach architectural concerns. |
| **17.** | Woods, 2015 | Search query 3 | Include, focused on collaboration problems of agile development teams and software architects |
| **18.** | Poort, 2014 | Search query 3 | Include, proposes advice that architects can help to become more effective in agile world without implementing new methods or frameworks. |
| **19.** | Diaz et al., 2014 | Search query 3 | Exclude, focused on specific framework and smart grids. |
| **20.** | Gill, 2014 | Search query 3 | Exclude, paper inaccessible. |
| **21.** | Lee and Baby, 2013 | Search query 3 | Exclude, too focused on fisk management. |
| **22.** | Zlatev et al., 2013 | Search query 3 | Exclude, paper design of an architecture for a specific application. |
| **23.** | Falessi et al., 2010 | Search query 3 | Include, discusses tension between agile and architecture communities, along with myths and facts about coexistence. |
| **24.** | Madison, 2010 | Search query 3 | Include, argues that agile development is not at odds with architecture. |
| **25.** | Gong (2012) | Manual | Include, proposes a conceptual architecture aimed at providing agility and flexibility. Example of case study approach in this field. |
| **26.** | Eloranta & Koskimies (2013) | Search query 3 | Exclude, paper inaccessible. |
| **27.** | Watfa & Kaddoumi (2021) | Search query 3 | Exclude, paper is a specific framework for Agile Enterprise Architecture . |
| **28.** | Alzoubi & Gill (2020) | Search query 3 | Exclude, about the effects of agile enterprise architecture on agile software development |
| **29.** | Gill (2015) | Search query 3 | Exclude, paper on agile enterprise modelling. |
| **30.** | Sandoval et al. (2017) | Search query 3 | Exclude, written in Spanish. |
| **31.** | Santos et al. (2021 | Search query 3 | Exclude, paper inaccessible. |
| **33.** | Mitsuyuki et al. (2017) | Search query 3 | Exclude, paper inaccessible. |
| **34.** | Bondar et al. (2017) | Search query 3 | Exclude, specific enterprise architecture framework is used to model a system-of-systems |

| | | | |
|---|---|---|---|
| | | | perspective. Unrelated to software development. |
| **35.** | Rane & Narvel (2021) | Search query 3 | Exclude, application of agility and Block-chain architecture in a domain unrelated to software development. |
| **36.** | Rane & Narvel (2021) | Search query 3 | Exclude, application of agility and Block-chain architecture in a domain unrelated to software development. |
| **37.** | Boyer & Mill (2011) | Search query 3 | Exclude, book chapter. |
| **38.** | Stoica et al. (2018) | Search query 3 | Exclude, conceptual design of an agile collaborative architecture for E-government services |
| **39.** | Bosch & Bosch-Sijtsema (2010) | Search query 3 | Exclude, book chapter |
| **40.** | Yli-Ojanperä et al. (2019) | Search query 3 | Exclude, paper about agile manufacturing concepts to the reference architecture model industry 4.0. Not about software development. |
| **41.** | Chicaiza et al. (2018) | Search query 3 | Exclude, paper written in Spanish. |

# Appendix B – Case study protocol

The purpose of the case study protocol is to define a set of guidelines that govern the case study research project (Yin, 2018). The case study protocol is contains a section on informed consent forms, case study questions. Normally, procedures, data collection, case selection and limitations are also included in the case study protocol, however these have already been included in chapter 3.

## B.1 Case study procedures

This section discusses the protocols, instruments and timeline that are used in the research.

### Data collection and protection of humans subjects

Yin (2018) identified six data collection methods for case study research: 1) Documentation; 2) Archival records; 3) Interviews; 4) Direct observation; 5) Participant observation; and 6) Physical artifacts. Both observation techniques are not compatible with the given timeframe of 20. The small timeframe this leaves for data collection. Thus it is very likely that the researcher will not be able to observe the consequences of governance strategies. A physical artifact (i.e. serious game) could be used to simulate governance strategies affecting (part of) a software development process in order to study the interaction of software architects and agile development teams, however time and knowledge to set up such an experiment are lacking. Mastery of the chosen data collection methods is identified as important by Yin. For this research expert interviews and document (including archival records if relevant) analysis are used as these data collection techniques are best understood by the researcher and this type of data is available to the researcher.

Since documents analysis and interviews are the chosen data collection methods, it is important to consider their benefits and limitations. Yin (2018) identified that interviews are prone to response bias and bias due to poorly articulated questions. Moreover, inaccuracies can arise as respondents have trouble recalling events and participants might say what the interviewer wants to hear (reflexivity). Documentation might be difficult to find or access due to i.e. privacy reasons. In analysing documentation a researcher should also consider the (unknown) bias of the author and biased selectivity. On the contrary there are also advantages to consider. Interviews can focus on the case study topics and provide insight in personal views as well as explanations. While document analysis is repeatable, unobtrusive, specific (i.e. on event) and can cover a long time span or consider many events or settings. Since availability of interviewees could be a limitation, a trade-off will have to be made between comparative capacity of cases and availability of interviewees.

Changes in governance are likely to be well documented and can support or contradict claims made by participants, on the other hand they might not be accessible due to confidentiality or privacy reasons. Therefore, this data collection method is considered a ‘nice to have’ and is not a hard requirement. Both software development practices involve the provision of documentation, however in varying degrees.

An idea booklet will be used to document thoughts and insights during the interview, this idea is based on Burgelman (1983) and is implemented as it is not always clear what will be important later on. For this reason and since the researcher is working alone, which makes it difficult to make extensive notes while also conducting the interview, the interviews will be recorded. The recordings will be deleted after transcription within 10 working days and validation by the participant. These transcriptions will later be summarised, once it is clear what is important and which details can be left out. The transcriptions will be deleted one month after the research has been completed. The main reason for this is to preserve the privacy of the participants and to protect participants and their employing organisations from reidentification through the research data.

Data will be collected, stored, processed and published based on informed consent by participants. An informed consent form (Appendix B.1) will be used to inform and acquire consent from participants on the research agenda, data storage procedures and time and publication of research data. Ranges are often used to present numerical data that will be published in order to decrease the risk of reidentification. TU Delft OneDrive will be used as a safe storage mechanism. The procedures

and forms have been reviewed and approved by the TU Delft Human Research Ethics committee.

### Research team

Another limitation is that only one researcher is able to carry out the data collection, even though Eisenhardt's (1989) methodology is meant to leverage the complementary insights of multiple researchers and increase the confidence in findings. However, the researcher is working under the supervision of a research team, discussions with this team can simulate the presence of multiple investigators with complementary insights and increase confidence in the findings.

### COVID-19

Face-to-face interviews are very unlikely to occur due to the COVID-19 pandemic. Since telephone interviews tend to lead to a less detailed response by interviewees, preference will be given to computer-mediated interviews (Irvine, 2011). Hence consideration need to be given to the differences between computer-mediated and face-to-face interviewing, such as informing the interviewee that a detailed response is desired and that there are no right or wrong answers when the interview questions are sent (Curasi, 2001). Moreover, since successes as well as failures will be the subject of this research, interviewees and organisations should be protected through anonymisation. Finally, the object of study (combination of people, approaches, methods, policies and tools) might change while under study, either by the fact that subjects are researched or for other reasons (Yin, n.d.).

### Time schedule and number of cases

Eisenhardt (1989) advises that adding cases and incrementing between theory and data should stop when theoretical saturation is reached. This is the point where incremental learning becomes marginal as the researcher starts to observe reoccurring phenomena. However, this point will be difficult to recognise in the research process as conducting the interviews, transcription and analysis of the interviews is likely to occur synchronously, depending on the availability of participants. To prevent issues with the time schedule hard deadlines have been set for data collection, transcription and analysis in Figure 13. This figure visualises the research activities and the amount of time they need to be completed. Figure 14 gives a visual representation of the Research flow and includes the same research activities as the Gantt chart. It connects the main research activities and deliverables. Data collection and transcription will both last 41 days, the latter starting and ending about a week later, while data analysis will last 46 days and ends last. Eisenhardt's advice of a number between four and ten case studies will be followed. Fewer than four case studies make it difficult to generate theory with much complexity and has unconvincing empirical grounding. While more than ten cases creates coping issues with complexity and the volume of data.

| Id | Research activity | Start date | End date | Duration (days) |
|---|---|---|---|---|
| 1 | Start | 31-1-2022 | 1-2-2022 | 2d |
| 2 | Meeting 31-jan | 31-1-2022 | 1-2-2022 | 2d |
| 3 | Finalise HREC application | 31-1-2022 | 14-2-2022 | 11d |
| 4 | Fine tune kick-off proposal | 31-1-2022 | 14-2-2022 | 11d |
| 5 | Case study design | 16-2-2022 | 23-2-2022 | 6d |
| 6 | Case study selection (criteria) | 23-2-2022 | 2-3-2022 | 6d |
| 7 | Prepare kick-off presentation | 3-3-2022 | 7-3-2022 | 3d |
| 8 | Kickoff | 8-3-2022 | 9-3-2022 | 2d |
| 9 | Process kickoff feedback | 8-3-2022 | 11-3-2022 | 4d |
| 10 | Literature review | 11-3-2022 | 18-3-2022 | 6d |
| 11 | Conduct case studies | 14-3-2022 | 9-5-2022 | 41d |
| 12 | Stop data collection | 9-5-2022 | 10-5-2022 | 2d |
| 13 | Transcript case studies | 21-3-2022 | 16-5-2022 | 41d |
| 14 | Stop transcription | 16-5-2022 | 17-5-2022 | 2d |
| 15 | Analyse case study data | 21-3-2022 | 23-5-2022 | 46d |
| 16 | Stop data analysis | 23-5-2022 | 24-5-2022 | 2d |
| 17 | Prepare Greenlight | 30-5-2022 | 6-6-2022 | 6d |
| 18 | Green Light meeting | 6-6-2022 | 7-6-2022 | 2d |
| 19 | Book a room for graduation | 9-5-2022 | 10-5-2022 | 2d |
| 20 | Revise report | 6-6-2022 | 20-6-2022 | 11d |
| 21 | Prepare defense | 27-6-2022 | 6-7-2022 | 8d |
| 22 | Defense | 4-7-2022 | 5-7-2022 | 2d |

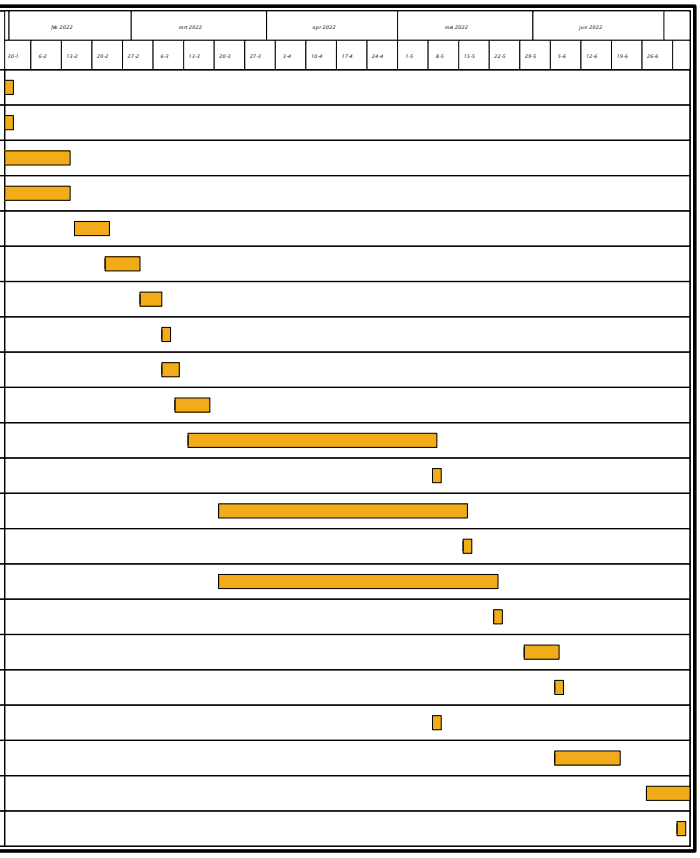


*Figure 13: Gantt chart of research activities*

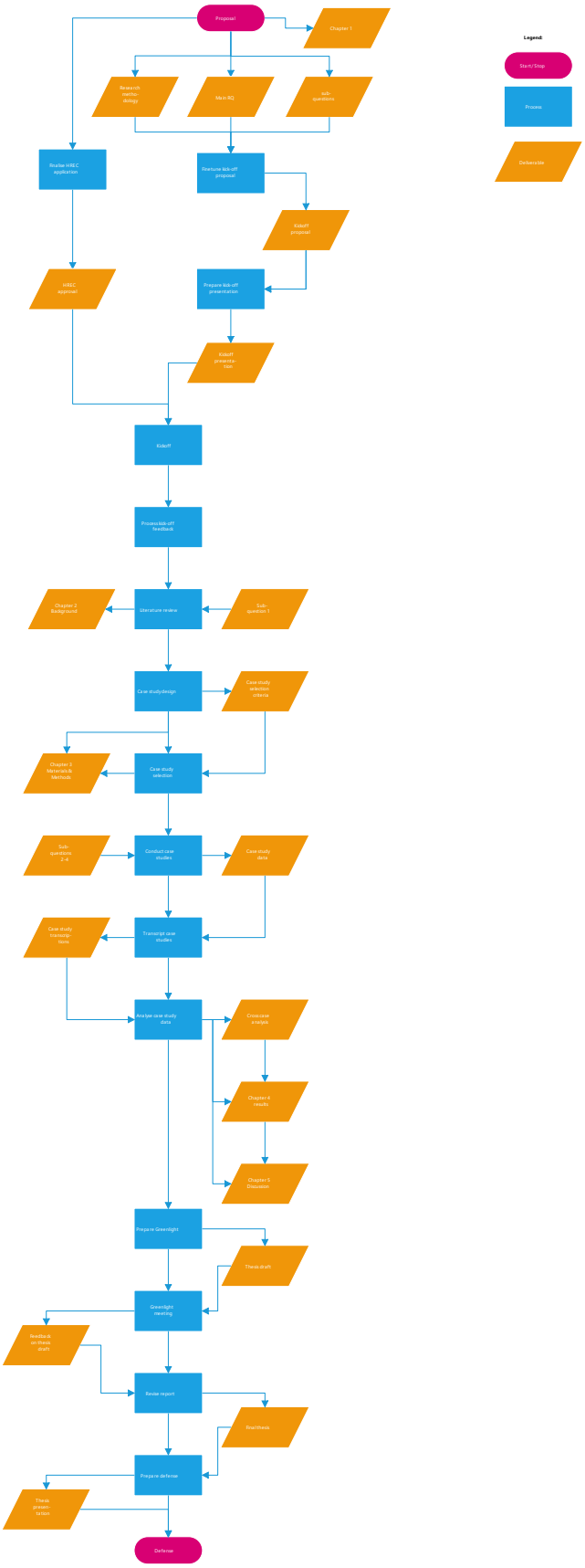


*Figure 14: Research Flow Diagram*

## Interview time

Interviews will take up at least one and no more than two hours, striving for one and a half hour. Less than one hour will not be enough to answer the full questionnaire. While interviews of more than two hours are considered to be disproportionally straining on the interviewees time. A limit is also useful for the researcher, while more data can seem better, it is useful to keep focus while collecting data. Additionally, since the interviews have to be transcribed, limited time will keep transcription manageable for the researcher. Especially, since at least two participants have to be interviewed per case, doing interviews that are longer than two hours can make the amount of data unmanageable.

Yin (2018) classifies interviews of under two hours as shorter case study interviews states that the open ended interviews can be conducted in a conversational manner. Similar to, prolonged case study interviews of 2 (or more) hours. However, he notes that the case study protocol is likely to be followed more closely. Especially if this type of interview will be used to corroborate specific findings, which very likely will be the case in the research due to the theory building character, the key is to ask questions in a genuinely uninformed way to allow the interviewee to provide a fresh commentary. If the questions are asked in a leading way, the corroboratory purpose cannot be served. Another way to do so, is by asking people who are known to have different perspectives, this is addressed by interviewing both an architect and agile developer, who are known for their different perspectives on software development. Consequently, questions are added in order to identify and understand the own interviewee's perceptions and sense of meaning on the material (Appendix B.2.2).

## Interview protocol

The interview protocol is presented in Appendix B.2.2. The interviews will be semi-structured with open ended questions to support rich qualitative data needed for theory-building. The interview questions might change as theory evolves or new insights emerge, changes will be documented to safeguard reproducibility and scientific rigor. Questions one until four are used to identify the context of the project, organisation of the product owner, role of the participant, expertise of the participant, agile adoption and architecture practices. Questions five until nine and thirteen until fifteen are related to governance, the interaction and added value and problems of the interaction.

While most of the data is of a qualitative nature, incurring phrases, classifications or descriptions from the participants, the closing questions on delivery of the project in terms of costs and savings or overrun, plus quality perception by the customer might result in quantitative data combined with a qualitative description, i.e. the project was 7 days overdue, because of reason X. As this information is useful to determine added value or incurred costs by governance strategies on the software development combination, this data will also be used if relevant. Similarly, the opening questions ask the participant about their experience in terms of years and will be considered in the following classification: < 5 years, 5-10 years, 10 – 15 years and 15 + years . Thus while the core of the research is of a qualitative nature, there are small quantitative elements that can add meaning or validity to findings.

## Supporting principles

Yin (2018) identified the following principles to address challenges in construct validity and reliability: 1) usage of multiple sources of evidence (triangulation); 2) creating a case study database; 3) maintaining a chain of evidence; and 4) exercising care in using data from electronic sources, i.e. social media.

The first principle is addressed by interviewing at least two experts that worked on a project and document analysis. Moreover, the individual cases will be subjected to cross-case analysis to increase the construct validity and reliability. A case study database that separates the raw data from interpretation will be used. The raw data should remain untouched after transcription and validation by the participant. This is important to preserve the integrity of the research and statements of the participants. The only exception to this of data and interpretation will be the summary that will be used

to provide transparency about the data that supports the findings while safeguarding the privacy of the participants. This means that updates in the form of notes should be made in the report and not in the raw data. Versioning will be used to increase transparency in changes of tabular materials, the case study report and master thesis. Documents that are studied are also to be added to the database and to be mentioned in the bibliography. New narrative compilations should also be part of the case study database. These are i.e. cross-references or classifications, interesting themes or ideas (basis for theory) and the researchers own open-ended answers to the case study protocol questions. The latter is part of the analysis and can be a starting point for the case study report and theory building process. The documentation of new narrative compilations and notes should not be edited extensively, but should be informative enough to connect the original questions of the case study protocol to the responses.

In order to establish a chain of evidence, the findings are to supported by a reference to the specific document or interview. The specific sources should be highlighted in this document and the database should describe the circumstances under which the data was collected. These circumstances should be consistent with the case study protocol. Thus the protocol questions and original study questions should be observably linked. However, some of the practices supporting this principle are at odds with the privacy that is guaranteed to the individuals, hence the aggregate summaries should be as informative and consistent with this principle as privacy allows. In short, privacy of the participants is chosen over this principle in case of conflict. The fourth principle mainly concerns interviews through chat-rooms, where it is not always clear who is on the other side. This is not relevant, as the computer-mediated interviews will take place via a videoconferencing software: Microsoft Teams. Additionally, the principle is for when social media posts from i.e. Facebook or Twitter are used. These sources will not be used.

## Field procedures

In theory-building from case studies there is frequent overlap in data collection and data analysis. Overlapping data analysis and data collection helps to take advantage of flexible data collection. As it allows the freedom to make adjustment to the data collection process, such as the interview questionnaire. This allows the researcher to take advantage of special opportunities that may arise or addition of data sources in selected cases. Additional cases or participants may be added that have become clear after data collection has started. Thus newly emerged lines of thinking can be discovered, which helps with theory building, as this new line of thinking can be tested. The researcher needs to be careful to document these new lines of thinking and changes to data collection to ensure that the approach remains systematic and documented.

According to Yin (2018) there are five desired attributes for a researcher conducting case study research: 1) ask good questions and interpret the answers fairly; 2) be a good listener and to think beyond existing ideologies or preconceptions; 3) stay adaptive and see new situations as opportunities, not threats; 4) have a firm grasp on the issues being studied and; 5) conduct research ethically, be open to contrary evidence. These desired attributes are in line with the idea of overlapping data analysis and collection and the use of literature in a theory-building case study by Eisenhardt (1989).

Asking good questions is important, as in case study research it is not readily predictable which information will become relevant, even though a formal protocol is followed. Thus in case study research, analysis occurs *during* the data collection process which is determined by the ability of the researcher to ask good questions and being a good listener. Being a good listener helps to understand not only what has been said, but also what has been meant.

In order to conduct case study research ethically and without preconceptions it is important to report contrary findings to two or three critical colleagues while still in data collection phase (Yin, 2018). This can help to offer alternative explanations and suggestions for data collection. The critical colleagues will be the members of the research committee in order to safeguard confidentiality. The other principles will also be implemented.

## B.2 Case study Instruments

### B.2.1. Informed consent form

**Information sheet for Consent Form**

The purpose of the research is to identify general governance strategies that can be used to achieve complementary added value form practicing software architecture together with agile software development. In this case, complementary means that the added value of combining both software development approaches is greater than the sum of individual added value of the approaches. To address this research topic, a multi-case study approach is used. In order to collect data for the multi-case study approach interviews with experts are used. Participating in this research will help researchers and practitioners to recognize potential added value and pitfalls of combining software architecture and agile development. The results will allow product owners, software architects and agile development teams to use the identified governance strategies to steer towards complementary added value in their software development process.

Participation is on a voluntary basis. This means that informed consent is needed from the participant and that this consent can be withdrawn at any time during or after the study, without having to give a reason. Participants are free to refuse to answer any question(s). Participants can withdraw from the interview by expressing their willingness to do so during the interview or later by sending a statement and referral to the subject of this research via e-mail: c.vandervliet@student.tudelft.nl. This e-mail address can also be used to file a complaint. Informed consent is needed separately for:

1) Participation in the research
2) The use of quotations from the interview in the master thesis
3) Publishing the findings from interviews master thesis and the anonymised summaries

Since the interview will be conducted by one researcher, video-recordings of the interview will be made. Being a participant in an interview comes with the risk of re-identification and/or reputational damage of the participant or the participants employing organisation. To address this risk the following mitigating measures are taken:

- The video-recordings will be transcribed in a privacy preserving way within 10 working days and send back to the participant, the video-recordings will be deleted after validation of the interview transcription by the participant. Participants have the right to view the transcription of their interview and ask for rectifications.
- The transcripts of the interview will be deleted one month after the research has been completed.
- Separate from the transcriptions of interviews, anonymised summaries will be created. Any data unrelated to the results will be deleted. Data that is relevant for the results and could potentially lead to re-identification will be generalised, using for example ranges instead of absolute numbers. Names will not be mentioned in the summaries.
- The recordings and transcriptions are only accessible to the researcher and graduation committee.
- Only the anonymised summaries and the master thesis will be made publicly available.
- Personal information about the participant such as names, email addresses, employers will only be stored for communication purposes in a secure institutional storage place and will be deleted one month after the research has been completed. This data will only be shared, if necessary, with the graduation committee.

It is important to know that a detailed response is desired by the researcher and that there are no right and wrong answers to the interview questions. By giving informed consent to participate in this research the participant acknowledges that they understand the information in this form, has be able to answer questions and received satisfactory results.

## Consent Form for: Agile software development and software architecture, complements or counterparts? The role of governance.

| *Please tick the appropriate boxes* | Yes | No |
|---|---|---|
| **1. Taking part in the study** | | |
| I have read and understood the study information or it has been read to me. I have been able to ask questions about the study and my questions have been answered to my satisfaction. I consent voluntarily to be a participant in this study and understand that I can refuse to answer questions and I can withdraw from the study at any time, without having to give a reason. | □ | □ |
| **2. Use of the information in the study** | | |
| I agree that my information can be quoted anonymously in research outputs. | □ | □ |
| **3. Future use and reuse of the information by others** | | |
| I give permission for the anonymised transcript summaries that I provide to be published alongside the master thesis in the TU Delft educational repository. | □ | □ |

**Signatures**

_____ _____ _____

Name of participant [printed] Signature Date

I have accurately read out the information sheet to the potential participant and, to the best of my ability, ensured that the participant understands to what they are freely consenting.

_____ _____ _____

Researcher name [printed] Signature Date

Study contact details for further information:
*Stan van der Vliet,*
*will add phone number later,*
*c.vandervliet@student.tudelft.nl*

### B.2.2 Interview questions

Share informed consent form with participant and obtain informed consent from participant.

Questionnaire

1) What is your current position at …. ?

2) I would like to talk about the software development process. Can you describe the software development process? Who was the product owner? Who was the end-user of the product? How would you define them?

3) What was your role in the project? What is your experience in this role? How would you describe the function of your role? What are the tasks and responsibilities? Which role in the process was responsible for flexibility? Which role in the software development process was responsible for stability?

4) How did you employ agile software development in the project? What was its role in the development process?

5) How did you employ (software) architecture in the project? What was the role of the (Software) architect in the development process? Did they design upfront? Was it possible to alter these designs?

6) How did agile software development and software architecture interact in the development process? What were advantages? What were disadvantages?

7) Now I would like to shift the interview towards added value or problems in the project. Could you describe added value or problems that occurred due to the interaction of agile software development with architecture?

8) Why do you think that this added value or problem can be attributed to the interaction of agile development and architecture? Why is it not related to either architecture or agile alone?

9) How did the agile team and software architect communicate? Why? How did this impact the project?

10) How would you describe the knowledge of the architect? / How would you describe the knowledge of the agile development team?

11) What types of uncertainties were addressed? How was uncertainty addressed in the project?

12) Let’s talk about governance and how it impacted the project. Could you describe what you consider a governance strategy?

13) Could you describe what you consider a governance strategy that affect your project with special attention to the interaction of SA and agile that we just discussed? Is this interaction affected by compliance to legislation? By budgets? Are you able to hire the right personnel? Or by tender-like procedures?

14) Could you describe in detail which added value or problems this delivered or incurred for your project due to this governance strategy?

15) Was the project delivered within allocated time and budget or was it not?

16) How was the project received by the customer in terms of quality? Was the quality of the project in line with what the clients' expectations?

17) Do you know somebody else who might be interesting to interview on this project? Are you able to share documents surrounding the events that we discussed?

# Appendix C – Case study report 0

Table that summarises findings of case study 0

*Table 16: Case study report 0*

| Variable name | Unit of measurement | Functional designer and Scrum Master |
|---|---|---|
| Project description | Description | Establishment of a nationwide application for the prevention of fraud. This system should support inspection and enforcement of compliance to rules. All three applications/uses of the application, registration, inspection and enforcement had different teams as different stakeholders were responsible for each part. |
| PO description | Tech savvy / non-tech savvy / description | One person from a ministry who delegated their tasks and responsibilities to a person from the organisations responsible for inspection and enforcement. So, there were two product owners that had the freedom and mandate to formulate requirements. I worked on the inspection team and thus collaborated with the PO from inspection. I was amazed at the PO's skills for this role, for example management of stakeholders or leadership but also leaving room for their thoughts on the product. PO was responsible for operation of other systems at their organisation, including the old system. So, somewhat tech-savvy, knew the problems and challenges of software(applications) and could translate what this means for operations. |
| PO impact on project | description | Both 'parts' were developed separately from each other and needed to work together. So, both POs would have to communicate well over things like architecture. Was difficult to engage PO at first, as their stance was: the PSA contains 200 requirements, so you guys could build right? However, after a lot of convincing the PO agreed to attend a three-hour workshop every week. |
| End-user description | Tech savvy / non-tech savvy / description | On the inspection side, the inspectors. On the enforcement side enforcers, which were all organised differently, so this part of the project was a bit more complicated. For example, a small organisation is organised differently than a bigger organisation. For this interviewee, the inspectors who had to go through the application and inspection process on the basis of the assessment framework (toetsingskader). The end-users were not so tech savvy, basic computer skills. |
| Role in project | Software architect, enterprise architect, member of agile development team, | Functional developer and Scrum Master |

| | | |
|---|---|---|
| Experience in role | < 5 years, 5-10 years, 10 – 15 years, 15 > years | < 5 years, agile was fairly new then and so was the Scrum master role. |
| Project status | Functional version / in operation phase / description | Reflection on a development process in the past. |
| Agile framework used | SAFe etc. | Three teams interacted with Scrum, SAFe did not yet exist. |
| (Reference) architecture framework | TOGAF, NORA etc. | NORA |
| Geographical scope | International, national, regional, local, central (same building/room) | Central |
| Agile implementation | Maturity / Description | Worked agile three months after project start. Before that: waterfall. After three months: made three teams: inspection, enforcement and a team that developed the nationwide application. |
| Role in project | Description | Used prototypes to get agreement on requirements during a three-hour workshop as PO's were not fulltime on site. During this workshop, product owners would bring relevant stakeholders on board. |
| Solution architecture implementation | Top-down, architect actively involved in agile development team and / or description | Architects went to work on a Project Start Architecture (PSA) at the start of project. This included assumptions and nationwide standards such as NORA. What do we do with data, where do we store it? How do we address functional and non-functional requirements? This was a Solution architecture+. System was developed from the PSA. The PSA was transformed iteratively into a Project End Architecture (PEA) which was transferred to the managing organisation. Architect that was interacted with could be identified as a solution architect. |
| Interaction between architecture and agile | Description | Functional developer/Scrum master and PO were involved in formulation of requirements for PSA.<br><br>Requirements from PSA were translated into features which were detailed in refinements, up to two sprints ahead.<br><br>Architect (PO, operations and other stakeholders) attended the workshops and was closely connected to the content. Workshops were used to discuss what was built and to discuss the next sprints.<br><br>Architect was asked each week to review user stories and deliverables. Architectural review was part of DoR. |
| Advantages or disadvantages of interaction | Description | Protypes and the ability to give immediate feedback helped to convince PO of importance of attending the workshops.<br><br>Packages that contained everything were delivered at the end of sprints, including documentation. Because of this one could access the right documentation and source code easily a year later. |

| | | |
|---|---|---|
| | | Uncertainty on requirements could was addressed by working iteratively and by showing prototypes to customers.<br><br>A good architect can be a gauge for the team to assess whether the right thing has been built. This role is not always carried out although there is a lot of added value in it. This allowed the team to build the right things on the first try.<br><br>A good architect is a visionary descriptor based on the organisation and the market. They look at how components should fit together.<br><br>An architect can be an interface to external stakeholders, for example operations.<br><br>Architect had good knowledge of government standards and interfaces that needed to be adhered to.<br><br>Good software architects should keep close ties with reality.<br><br>Frequent interaction and approachability of software/solution architect can help to avoid the development team getting lost in the architecture, while also avoiding the pitfall of losing oversight in short cyclical agile working methodology.<br><br>Reduced the disadvantage of a purely waterfall architecture approach of straining the development team too much with upfront design, as in software development you cannot know everything beforehand.<br><br>Technical uncertainty could be approached stepwise in an agile approach together with the architect. Architect played a big role in this.<br><br>An architect has a lot of added value towards a products owner, as the PO has a functional perspective, while a good architect can help a PO to think about non-functional requirements. |
| Interaction specific problems, tensions or bottlenecks | Description | Agile working methodology needs to be supported by the architecture. The architecture and principles should enable the teams to develop autonomously and independently.<br><br>Scaling of agile teams is difficult; teams need to be tuned into each other if they work on one product.<br><br>For smaller systems it may seem like an architect can design everything upfront, however issues |

| | | |
|---|---|---|
| | | arise if these systems need to be scaled.<br><br>An architect needs good communicative skills and feeling for the business, but also good technical skills to talk to developers. This makes them difficult to find and hire.<br><br>Often architecture roles are separated: business architecture, solution architecture, domain architecture, however an architect should be able to look across these boundaries, otherwise it will be impossible to integrate these aspects.<br><br>Tendency of architects to make things too standard, which can reduce the ability to adapt things later on.<br><br>Waterfall approach can lead to a lot of rework even though a lot of time and effort has been put in, in the early stages.<br><br>Risk of too much attention to here and now and too little attention for the future.<br><br>Architect has to have knowledge of agile working methodology, understanding on what that entails is needed.<br><br>Bottleneck: architect has to be open to working in agile environment. If architect has a more traditional perspective on software development or a more layered architecture organisation, the interaction might not work well. |
| Communication in combination | Irregularly / weekly / daily / Description | Architect stood close to team and was easily approachable. |
| Impact of communication | Description | Differs from other teams that claimed to be more agile, was better interaction between architect and agile team in this case than in others. |
| Knowledge of architect / team | Description | Good. |
| Uncertainty addressed (types of) | Costs / schedule / quality / description | Low uncertainty in requirements on application and inspection as there was an old system. For enforcement, this was not the case.<br><br>Technology, amount of load that needed to be handled.<br><br>Uncertainty on specs of from other organisation. |
| Impact of uncertainty on project. | | Challenging technical architecture on non-functional part. Architect was responsible for this. |
| Strategies to address uncertainty | Description | Agile methodology helped to address high uncertainty for what should be built for the enforcement system through prototyping with stakeholders. Determining how much points you can work on in the given timeframe helped to |

| | | |
|---|---|---|
| | | engage in discussions on what should be developed and what not.<br><br>Approached technical uncertainty in a stepwise approach together with architect. Architect had a big role in this. |
| governance structure | Description | 3 teams with a PRINCE 2 'hood' on top, but agile below this.<br><br>Architect was shared over two teams.<br><br>Operations used 'throw it over the fence model.' |
| Impact of governance structure on project | Description | PRINCE 2 to outside, risk log etc. Agile procedures within teams helped to manage risk, I.e. the sprint-rhythm, realising, DoD, being transparent on the progress.<br><br>Interaction was centralised, which helped as the architect could keep overview. Responsibility for interaction with external stakeholders was also centralised (as it was put with the architect).<br><br>Governance of operations was incompatible with agile governance, as they came with a long list of operational requirements instead of formulating them together. |
| Timing of event | Description | After 3 months. |
| Management | Description | PO was good in managing the stakeholder group.<br><br>Product owner got really good at prioritising after a few months. |
| Impact of management on project | Description | End-users were present in the workshops to give feedback.<br><br>PO communicated to other stakeholders that trade-offs needed to be made, while still leaving room for feedback. This is what you expect from a PO. This was a very good execution of the PO role. |
| (Changes in) procedures | Description | Started working agile after 3 months. |
| Impact of procedures on project | Description | Less uncertainty on requirements, more stakeholder support. |
| Project delivered within budget | Yes /no + description | Yes, new budget asked and approved for increased functionality. |
| Project delivered within time schedule | Yes /no + description | Yes, delivered within 12 months. Extension for increased functionality. Transferred to operations after 12 months and short pilot. |
| Project delivered within client quality expectations | Yes /no + description | After a year, the system was further developed while it was in operation. |

# Appendix D – Case study report 1

Table that summarises findings of case study 1.

*Table 17: Case study report 1*

| Variable name | Unit of measurement | Back-end developer | Solution architect/team lead |
|---|---|---|---|
| Project description | Description | Software application where lot of new data needs arose during the development process.<br><br>Crisis structure which caused ad hoc requirement(s) (changes). | Software application with internal and external stakeholders. Very political project with a lot of stakeholders. Stakeholders were often unwilling to commit or give concrete answers, wanting to be able to frame things differently if needed, which caused difficulty for the teams that needed concrete decisions in order to move forward with the technical product. |
| PO description | Tech savvy / non-tech savvy / description | MT consisted of two program managers who determined what would happen, they were in contact with the minister and thus operated as PO.<br><br>Contact was frequent, PO's stood close, however participant has experienced cases where the PO stood closer to the team. The PO's did attend the dailies with the whole programme. This had to do with the fact that there were several teams, not only development teams.<br><br>PO's did have a lot of expertise on policy, scientific affairs and the use case that the application was developed for. | Multiple, not tech-savvy / never have led a technical project before, policy makers, not clear on requirements and their urgency, very involved, political career over technical product |
| PO impact on project | description | PO's were both non-tech savvy and had no clue on the technique behind the software application. But did have a clear opinion on the software application and what should happen with it. | Interests to superiors had more weight than delivery of a good technical product. There was no focus on the long-term of the technical product.<br>Urgent tickets jeopardized the stability of the technical product and of the development process.<br><br>Unclarity on requirements and their urgency led to mistakes, errors, lagging behind of documentation and technical debt. High involvement in combination with no technical background led to delays. Calls in the weekends and changes in requirements after sprint planning put extra stress on teams.<br><br>Unclear requirements made it difficult for development team to work according to sprintplanning.<br><br>Lack of recognition for quality attributes from PO's build technical debt and pushed issues to overtime. |
| End-user description | Tech savvy / non-tech savvy / description | Non-tech savvy | Not tech-savvy, illiterate, end-users could raise questions about the software application and were critical in doing so. |
| Role in project | Software architect, enterprise architect, member of agile development team, | Back-end developer, develop systems that collect and transform data and send data to the front-end. Continuous optimisations to | Solution architect and team lead (cloud engineer, software engineer, data engineer, BI specialist, operations/administrator of system, |

| | | | |
|---|---|---|---|
| | | develop the software application further. | team manager Jack of all trades), not suitable for one box, can be summarised as cooperative foreman. As team lead: Coordinating towards team. Discuss priorities and possibilities with PO (if PO comes with requirements after sprint planning). Keep team out of the heat.<br><br>As solution architect: make designs/drawings of how architecture, cloud architecture and data streams should be, so that the team can develop those elements. But also implement these things myself. Devise the frame in which can be worked, keep tabs on how things are going.<br><br>Operations: keep system running and functional (dataflows, KPI's, calculations in the right way etc).<br><br>Communication with internal team members and communication with other teams.<br><br>Development tasks.<br><br>Final responsibility. |
| Experience in role | < 5 years, 5-10 years, 10 – 15 years, 15 > years | < 5 years | 5-10 years, first time in political playing field |
| Project status | Functional version / in operation phase / description | Operational | Operational |
| Agile framework used | SAFe etc. | None. There might have been thought about using a specific framework, however in reality it was not possible to say, that was this framework. | None |
| (Reference) architecture framework | TOGAF, NORA etc. | Not discussed | Not discussed |
| Geographical scope | International, national, regional, local, central (same building/room) | Same room crisis structure | Same room crisis structure |
| Agile implementation | Maturity / Description | Front- and back-end team worked in synchronised sprints of two weeks. First week was mainly for building, second week for integrations, testing, releasing and cleaning. Non-development teams did attend sprint-demo and dailies. | Development work was assigned to developers through tickets via a backlog by the PO and team. Team-members gained responsibilities over time like releasing etc. Took a while for team was able to carry out activities like this by themselves. Refinements was done by another team and never done in time / at a satisfactory level. |
| Role in project | Description | Working agile helped to keep things organised. People know what the others are doing, it especially helped to align the front-end and back-end. Participant stated that they did not know better than to work agile with a team to develop a product. | Used to pose deadline, participants don't think agile was well implemented. This became better after a PO was introduced with experience on leading technical projects. |
| Solution architecture implementation | Top-down, architect actively involved in agile development team and / or description | Architect built tooling and used various components for this. These components were used by the developers in daily activities.<br><br>Architect delivered design and implementation of database, components and connections with customer portal. | Architecture was barely implemented, one person set up the data warehouse and data orchestration in the first week. After that architecture was not recognised by participant as a role that was employed. However, participant did take responsibility for quality attributes after further inquiry. |
| Interaction between architecture and agile | Description | If a component in the tooling did not work properly or we encountered problems, we would | Since PO did not recognise the backend (things that happened beyond the website) the agile |

| | | | |
|---|---|---|---|
| | | call the team lead/architect and then he would rethink and change things.<br><br>If new requirements occurred that was not supported by the current tooling, the architect would make sure that the tooling could support the new requirement.<br><br>Architectural items were present in sprint backlog and implemented through sprints.<br><br>If architecture changed, the team needed to alter their way of working. For this the team needed clarity on what has changed. | process only served to address the website. Thus, the architecture-agile interaction was not present.<br><br>Because of the combination of the role on the one hand the Sol. Arch/Team lead is making architecture decisions and designing the architecture. For example, how the dataflows should go. The team will execute this, but the Team lead/sol. Arch. Helps with this as an engineer. So, there was no real separation in this. |
| Advantages of interaction | Description | Fast communication<br><br>Architect role was the connection point with the infra team.<br><br>If you get a new request that you cannot address with the current solution, you can work on your architecture in sprints so that the new solution can be supported. Thus, the architect built or enabled new functionalities for the development team and the interaction of architect and team speed up the process of implementing the new functionality.<br><br>The interaction with an architect allows the team to do more than when it would have just been developers. A team that has an architect can address issues that are more out-of-the-box.<br><br>An architect that is part of the development team can be useful if the architect can do developer tasks as well, as this reduces overhead if there is a sprint with no or little architectural work to be done. | No need to account for architecture decisions.<br><br>We were sharp, had eyes on the ball and were always able to publish anything in time. We did not make any extreme mistakes that were traceable to us. Mistakes that occurred were due to other parties not delivering the right data, as we had to trust other parties to deliver the right data. |
| Disadvantages of interaction | Description | An architecture function of often scarce, I can image that putting the architect in an agile team can result in the architect being approached for issues unrelated to architecture.<br><br>Overhead, there are sprints where an architect cannot do much, this can induce overhead. However, this was not experienced in this case as the architect could also carry out developer tasks.<br><br>Architect in agile context gets approached with issues that are unrelated to architecture. | Frequent crashes occurred due to technical debt, requiring hot-fixes. Resulting in overtime for the team. |
| Communication in combination | Irregularly / weekly / daily / Description | Dailies and other scrum meetings. Calls throughout the day, easy to reach other. | Solution architect was part of team, thus daily communication through the daily stand-ups. Additional interaction in the sprint planning sessions and team building activities. |
| Impact of communication | Description | Get things done faster. On the other hand it could be more | People were able to find each other easily. Team building helped to |

| | | | |
|---|---|---|---|
| | | pleasant for the architect to work less close to the fire and experience less of the everyday affairs. This would result in fewer requests, since the distance is bigger. | algin and understand each other's characters.<br><br>It is important to trust each other within the team, that people dare to communicate openly and are not afraid of another's opinion. Also, towards the team lead. |
| Knowledge of architect / team | Description | Good, experienced, positive impact on the project. Could make the right choices. | Team: only knowledge of SQL. While knowledge of Java and C# would have been helpful. Knowledge about Git, branching strategies, pipelines, releasing etc was also limited at start. This grew over the project.<br><br>Lack of technical background for PO's. |
| Impact on project of knowledge | Description | Architect allowed to make the right decisions. | Technical knowledge of PO would have helped to identify that components need maintenance. Because there was no knowledge of this, this was not included in the process. |
| Uncertainty addressed (types of) | Costs / schedule / quality / description | Requirements: last minute changes in the course. | Requirements: A lot of urgent requests/tickets after the sprint planning.<br><br>Sprint planning and prioritisation changed continuously during sprint due to political nature of project.<br><br>Tickets were badly defined by responsible stakeholders and never really finished on time for sprint planning.<br><br>Staffing: Do I have enough people?<br><br>Technical: do the data streams flow right? Will I get my data? What will be the quality of the data?<br><br>Front-end web application was hosted by an external party which used Dockers, while these were not suited. The server's memory could not handle this, and this would cause long deployment times and frequent outages. As these also occurred on weekends or holidays this put further stress on the team, who would have to find the issue and rebuild the pipeline. As the system administration party was not really suited for this role.<br><br>There was only one person who could program in C#, so what if this person became unavailable? Who will check their work? Errors due to this could cause outages. |
| Impact of uncertainty on project. | | Lot of redundant work on features that were not needed after a week or that were not put live on the production environment.<br><br>Requirements entered after the sprint has started in the last month. Additional requirements would then be forced into the sprint on Monday or Tuesday, which then turned out to be not necessary on Wednesday, but on | Requirements: Urgent tickets disturbed the flow of the sprint and reduced time that could be spent on the actual sprint goals and quality attributes in working hours. Resulting in lower code quality, documentation lagging behind and affecting the stability of the project becomes endangered.<br><br>Sprint-planning was often thrown overboard after 1,5 days. |

| | | | |
|---|---|---|---|
| | | Thursday morning it appears to be necessary, on Thursday afternoon it is necessary in a different way than before. This all happens when the team wants to be ready on Friday to start testing and releasing. | Difficulties for team to implement tickets as stakeholders did not commit to their decisions.<br><br>Team would realise that a ticket is wrong, have to ask questions, revise work/tickets which resulted in delays and overtime as tickets needed to be finished in that sprint.<br><br>Staffing: time spent on internal politics.<br><br>Technical: issues with data are mainly caused by other stakeholders, however, can result in overtime for back-end dev. team. |
| Strategies to address uncertainty | Description | Changes like this are inherent to the process and have to be included if they come from an important figure in the government.<br><br>Try to signal whether the requirement could be implemented as early as possible.<br><br>Be clear if the ticket needs further refinement. If we found out at day two then we missed information, we should not have accepted the ticket. Therefore, we were quite critical in the refinement sessions to avoid nasty surprises.<br><br>In sprint planning we communicated clearly what we could handle and if new requirements came up, we referred to the sprint planning to show how the new requirement affects the sprint planning to make trade-offs transparent in respect to when it would be possible to release. | The team lead pushed back later in the project on urgent tickets and communicated that for this ticket to be addressed others would have to be dropped and a decision about this would have to be made. However, since quality attributes had no priority with PO's this strategy did not solve the technical debt.<br><br>Urgencies were further addressed by reserving time for urgent tickets.<br><br>Staffing: finding senior leadership support for resource allocation.<br><br>A front was created between the front- and backend to draw a line on tickets that were not well refined. These were sent back, which caused the responsible stakeholder to plan more refinement sessions and increased ticket quality.<br><br>For the hosting part, transparently communicating the consequences towards the client if this setup continuous to exist. Administration/management of the application was transferred to another party who put it in the cloud. |
| Governance structure/strategies | Yes /no + description | Back-end development team did not directly into contact with other teams such as communication and policy teams (outside dailies with the whole programme and demo's).<br><br>Other teams did not work agile, but more ad-hoc.<br><br>Separate infrastructure team.<br><br>Crisis and layered structure. Implementation of new legislation occurred during the project that required changes to the application. | For the party that was responsible for the data delivery, one team was made responsible to address data issues for this project. Previously there was no single responsible person, which caused issues in approaching the right person within this organisation if there were issues with the data.<br><br>Government entities had clearly demarked responsibilities. |
| Impact of governance structure on project | Description | Because other teams were present in agile rituals no misunderstandings occurred, a sort of mixed-from emerged.<br><br>This worked fine as agile is for software/ICT related activities, | Single responsible team within organisation responsible for data helped, as this person would address the issue internally.<br><br>Stakeholder form policy team was also program manager, this should |

| | | | |
|---|---|---|---|
| | | while operations has to keep things running and issues, which is more ad-hoc work, being less suited for the agile approach.<br><br>Normally this infrastructure is the responsibility of the architect, in this case the architect could serve as an interaction point, since he could join in on this conversation.<br><br>Ad hoc requests for new requirements or changes in requirements. Requirements came in through PO but have been by a few teams before they reach the developers. | have been separated. If there would have been a project manager, there would have been a clearer separation, somebody who could push back. This should have happened if you take the perspective of the product.<br><br>As the project was very political, the chance of errors increased.<br><br>Other entities that project was dependent on were not always cooperative. This also had to do with understaffing. |
| Timing of event | Description | Not discussed | ¾ years in. |
| (Changes in) management | Description | The PO changed during the project in the last month that participant was on the project.<br><br>Additional staffing was allocated to the project during development, as the backlog and sprints were getting fuller. | Yes, PO changed from non-technical to more technical person + manager of dev. team was out of running for a while.<br><br>Over time the team lead learned which people did raise issues in time and who did not.<br><br>High workload and short time schedule created high workload for team. Teams were all addressed for issues in the product.<br><br>PO's were confusing and micro-managing.<br><br>PO's prioritised new functionalities based on internal politics higher than structural improvements in non-functionals.<br><br>Manager of back-end team protected the team from last minute requirement changes or new requirements, by showing that not everything could be done. In other words, by making the trade-offs visible.<br><br>Lots of ego's involved in the project. |
| Impact of management on project | Description | Not discussed. | Would have been less stressful if the project management would have been more professional and experienced with technical projects. This should have been somebody who had zero interest in the political arena. The different interests played a role, for example the fact that one slip can be fatal in the political arena or that the project management has ambitions for a political career. In this case the head of the policy team was the program manager. There should have been a separate policy team with a senior stakeholder that communicates with a project manager would have resulted in a purer line of distinction. This would have allowed the project manager to push back if things would not fit the current planning. Which would have been better from a product point of view, as now there was a |

| | | | |
|---|---|---|---|
| | | | stakeholder that had a predominating interest, making their will the law.<br><br>Ability to educate the people on when to raise issues. Management style of team lead changed from top-down to laissez faire as development team became more experienced.<br><br>Degrees in priority requirements that needed to be added after the sprint planning. While micromanager delayed business. This caused people from other teams to drop out early of the project. Confusion occurred due to miscommunication.<br><br>High workload and short time schedule which increased the number of errors made. Thus, increasing technical debt. Moreover, it required a lot of the teams, leading to unhappy people that did not deliver to their abilities.<br><br>Decreased stability of the application. Resulting in more work for developers that had to do fix issues, while these issues could have been prevented by more attention to non-functional requirements.<br><br>Team was able to maintain stability, while other teams did not. Manager took over the role PO. Which was an additional role next to team lead/Sol. Arch. Pushing back helped to see through which were important changes or new requirements and which were not.<br><br>Project had the image of being the pet project of someone important rather than for the public good. Which affected the cooperativeness of other stakeholders. |
| (Changes in) procedures | Description | Constantly seeking for ways to put information of refinements in the team. How can we ensure that everybody understands the code that has been written through documentation? A wiki was used to document the way of working.<br><br>Refinements were organised with one or some members of every team, there was a club of different teams that refined the tickets.<br><br>Documentation was used to share knowledge on changes in the way of working through architecture changes. | Due to inexperience of team, team lead was involved in a lot of aspects of the projects such as releasing. As the team gained experience this decreased. This took about 4 months with the new team.<br><br>There was no time allocated for peer reviews and there was nobody who could do reviews of certain pieces of codes: mainly architectural parts. |
| Impact of procedures on project | Description | Way of working changes slowly over time, this was documented to share knowledge within the team.<br><br>This caused issues as person A within a team could have refined | Additional work for team lead.<br><br>Increased occurrence of bugs. Decreased stability of the application. |

| | | | |
|---|---|---|---|
| | | the ticket, while person B needed to implement the ticket. Tried to resolve by presenting all info on the wiki but caused more overhead than added value. Then tickets were assigned in such a way that members worked on the tickets they had refined themselves or had expertise in. Short calls also helped to share knowledge. | |
| Project delivered within budget | Yes /no + description | Extra staffing was added because the backlog filled up.<br><br>Were put on the project for the duration of the project so difficult to say if we could go out of budget. | Yes, budget was no issue since it was a crisis. |
| Project delivered within time schedule | Yes /no + description | Project was prolonged, thus extra budget was also granted. | Yes, team never missed a deadline. |
| Project delivered within client quality expectations | Yes /no + description | No issues, as it was clearly defined what should have been delivered. Back-end development is simple, right or wrong, other than front-end a discrepancy can occur on what has been designed and delivered. Still, this was also mainly in line with expectations due to the sprint demos. | Yes, Even though difficulties were encountered in this case, the team succeeded in delivering quality on time. Client gave a very high rating on quality afterwards. |

# Appendix E – Case study report 2

Table that summarises findings of case study 2.

*Table 18: Case study report 2*

| Variable name | Unit of measurement | Back-end developer | Team lead/Solution architect |
|---|---|---|---|
| Project description | Description | Authentication platform migration from old version to new version. Ongoing project in operational phase. There are one or two screens behind which happens a lot of magic. | Authentication platform migration from old version to new version. Ongoing project in operational phase. The platform is mainly technical, there is only a small functional area which you can see. Very large project with a lot of code, some of which might reaper after multiple years, so quality is very important. |
| PO description | Tech savvy / non-tech savvy / description | PO does understand functional added value of technicalities, such increased robustness or security. However, cannot understand the lines of code, but can bridge technology and functional added value. | Functional PO is complemented by a more technical role. |
| PO impact on project | description | PO is internal from the organisation that is responsible for the platform.<br><br>PO does the organisation of the service on a larger scale, communicates with external parties, sometimes technical parties that we need to include in the workshops and discussions. PO also does communication on the website and organises a beta-test with various modules. Feedback that comes out of this enters the backlog through the PO. | Key towards organisation and external parties.<br><br>PO chairs most Scrum ceremonies and prepares the refinements. |
| End-user description | Tech savvy / non-tech savvy / description | Different end-users since it is a platform, both tech savvy and non-tech savvy end-users. | Different end-users since it is a platform, both tech savvy and non-tech savvy end-users. |
| Team structure | Description | Implementation Engineer checks off items as Done. Everybody does testing in the team.<br><br>Operations team member that works mainly on releases of new versions of our software and packages and deploys them on the test, staging and production environment. This role also makes connections and configurations needed for applications to interact.<br><br>Front-end themes are done by designers. They should allow other parties to do self-service and administer their connections.<br><br>Three back-end developers, one of which is team lead/solution architect. | Team is layered. DevOps engineer and implementation engineer. The latter is the technical interaction point, i.e., what are the limitations of components? Both work for a project manager who is the key between development team, the organisation and external environment and collects requirements of external parties. The project manager has a more functional focus, for example on the flow and text in the application. Thus, the project manager and implementation engineer support each other, one being more technical, the other more functional. This can occur as the Scrum team is very open.<br><br>Testing capacity was already present in the team. |
| Role of interviewee in project | Software architect, enterprise architect, member of agile development team, | Back-end developer, implement back-log items, think along in functional issues from the business and give a technical solution for this. Delivery of quality and support. | Solution architect. Started out as developer/Scrum master and later replaced the former solution architect, which left.<br><br>End responsible for what is delivered, as team lead and back- |

| | | | |
|---|---|---|---|
| | | | end team are hired by the organisation that owns the platform.<br><br>Supporting and training the developers.<br><br>Shadow Scrum master, coach team members in background to raise issues at the right time. |
| Experience in role | < 5 years, 5-10 years, 10 – 15 years, 15 > years | < 5 years | 10 – 15 years |
| Project status | Functional version / in operation phase / description | Operational phase | Operational phase |
| Agile framework used | SAFe etc. | Scrum with own alterations | WaterScrum |
| (Reference) architecture framework | TOGAF, NORA etc. | Not discussed | Not discussed |
| Geographical scope | International, national, regional, local, central (same building/room) | Remote, but with offline same room meetings for architecture workshops. | Remote, but with offline same room meetings for architecture workshops. |
| Agile implementation | Maturity / Description | Mature, Scrum ceremonies. No need for Scrum master anymore as team members trust each other and know the process.<br><br>Agile serves to define, priorities and document items on our to do list. | Mature, Scrum ceremonies. No need for Scrum master anymore as team members trust each other and know the process. 2 week sprints. |
| Solution architecture implementation | Top-down, architect actively involved in agile development team and / or description | Architecture workshops on how applications and data streams should come together in the landscape with all stakeholders present in which everybody's input is valued.<br><br>Devs also work on architecture items in backlog. Architect is Senior Dev and team lead with formal architecture role. | Solution architect is hands-on (writes codes, starts up applications, delivers packages to DevOps). Architect thinks about bigger picture and involves stakeholders in open discussions about this. Creates a speak-up culture in which issues and questions are important.<br><br>Solution architect discusses a lot of architectural issues such as scalability, redundancy etc. with the implementation engineer and DevOps engineer, as they are responsible for the technology and infrastructure, making them perfect to design and setup the technical environment. This was done in formal and informal sessions.<br><br>Solution architect made a large sheet with the outcomes of the offline landscape workshops.<br><br>Leading in the development process, if things are to be developed, I would like that to happen in a certain way.<br><br>I often build completely new services; however, I try to distribute certain tasks to the other developers. For example, parts or large parts of applications. This decision needs to be supported by the team. |
| Enterprise architecture implementation | Description | Not discussed | There is an enterprise architect in the organisation, however this person is more involved in the chain. They can tell you what it does, but not on a technical level. |
| Interaction between architecture and agile | Description | Architecture and agile feed each other. Agile is process of working: formulation of backlog items, prioritisation, and | Distinction between design and implementation phase to unburden the team as there is a hard deadline to go live. For every step in the |

| | | | |
|---|---|---|---|
| | | implementation. While the architecture feeds the backlog.<br><br>Architect is a member of the team and develops alongside with the team.<br><br>Each team member can work on a piece of architecture, as architectural requirements enter the backlog as requests that are addressed in a user story.<br><br>Architecture feeds the backlog and agile structures the implementation of architecture work. Each practice feeds the other. As, the agile process says little on the content of the requirements or software development. It determines the way we work. But on what we work, it has little say on. The architecture and business provide us with content. So, the combination ensures that we can deliver what we deliver and loans itself for advantages like responsiveness and adaptability. But on implementation it does not add value if you don't know what to change. | process, a design is prepared by the project manager, implementation engineer, designers and sometimes an additional role. These designs together with the whole flow are presented to the development team. The development team can then raise questions about the flow or implementation order, possible improvements or the ability to implement at all, but also whether the work can be timeboxed. The solution architect has the biggest role in this, as he was involved in the old platform and remembers which choices were made there and why. This session is held weekly. This is an hour-long refinement session for design stories.<br><br>An additional hour is planned for refinement of technical stories.<br><br>Offline architecture workshop with the whole team in the same room in which the landscape was discussed. Which applications are there? What are the connection between the distinct applications? Which applications are involved in which parts of processes and how does the data flow? Which functionalities do we miss? Sort gap analysis and to get an overview of what is out there, what are we talking about? From bird-view to details.<br><br>Best place to discuss changes in functional requirements are in the refinements of the agile process. As the impact of the change is discussed here on other parts of the platform.<br><br>I think that the agile component is stronger than the architecture component. If you are an architecture-fanatic, then you work out a complete architecture on a basis of the modules, and put it down as given for how it's going to be built. We are more in dialogue. We develop a certain way to accommodate standardisation as much as possible, this way the architecture evolved over time. But since we are in a learning environment, everybody knows how the platform is built up in some degree, also the DevOps engineer and designers. And because we are in a learning environment, you should not prescribe things, you need to facilitate discussion and give back things. You have to give a starting point to develop into a certain architecture. Start with a blueprint (architecture) with main elements, but not everything yet. Some holes in it. Fill these holes in discussions with developers, operations, and business. Then apply Scrum on the blueprint and outcomes of the discussions. What our developers |

| | | | |
|---|---|---|---|
| | | | do really well, is working out their ideas and to discuss these with stakeholders before they start developing to test their assumptions.<br><br>The architecture is relatively agile. If it would have been waterfall, then all the boxes would have been coloured. We define a structure with some questions marks, that can be filled in later. What helps is a modular architecture, as much micro-services as possible, as many API's as possible, as much pluggable as possible. To remain flexible in that way, so that if something needs to be changed, not the whole application structure needs to be changed.<br><br>Also work with layered architecture, the back-end and front-faced applications are different layers. The front-faced applications are publicly accessible. For applications that have a front- and back-end application in one application there is also some layering. A piece of layered authorisation is also present. |
| Advantages of interaction | Description | Work process is experienced as good by participant.<br><br>It helps to have all requirements in an early stage and to have the ability to discuss them. This allows us to identify and address problems that we can already foresee. This relieves the problem of having to start over again because you found that something does not fulfil.<br><br>Internally, everybody is up to date and we have an open and direct culture in which stakeholders (operations and business) can share their thoughts. This reduces nasty surprises when we deliver software, as all involved stakeholders are on the same page. I.e., this makes that operations can actually manage the implementation that is developed, as they have the ability to state their concerns.<br><br>We are good at dealing with change. Ability to address production issues fast which provides added value for PO and customers. New requirements or issues are discussed in the regularly held refinement sessions. If we cannot make an estimate based on this, then we push it to the next refinement. Things that are identified on the backlog are given technical interpretation and prioritised. This allows us to respond to change and is added value for the PO and client as well. | Refinement of designs with whole team early on helps to avoid finding out late that you are building the wrong thing.<br><br>Offline architecture workshops helped for developers as people could share their experience with applications they touched and see everything together all of a sudden. This gives context for individual applications, shows what the platform is dependent on and how it interacts. This background is needed to develop things in the right way.<br><br>If a story cannot be implemented within the sprint, it is known for everybody. It is not a surprise, as it has been discussed during the daily stand-ups whenever a sprint cannot be completed.<br><br>Retrospectives each sprint help to improve ourselves continuously.<br><br>Research story for uncertainty helps to make it tangible and measurable. You can timebox and assign points to it, document it in JIRA and show what you did in the allocated time. This helps to relieve time pressure, as you don't run into the issue that you need to do research, but the sprint also needs to be finished. It becomes an element in the sprint, and you award yourself the time to do research in this way.<br><br>The way of working is efficient for the current process. There are |

| | | | |
|---|---|---|---|
| | | Since everybody is up to date, relevant stakeholders have the ability to be transparent to and discuss issues with their superiors.<br><br>Architect as senior developer within the teams makes him approachable for issues that (more) juniors run into.<br><br>There is a risk that the envisioned sprint planning goals are not met, as reality required the team to adapt. However, the PO understands this, as they know why this is.<br><br>No official clearance needed to make changes due to trust in team. This allows the team to anticipate issues they see looming on their path. | certain design structures that can be reused, and I know where to find them. I can find and use them quicker. I also know how to combine them with modern techniques.<br><br>The fact that the ownership for the architecture role is with one person helps to establish the role more strongly.<br><br>No large technical debt in backlog. Although there is technical debt in applications.<br><br>Processes (I.e., peer reviews) ensure quality of code delivered and increased learning for developers. Developers learn from each other, through peer reviews, but also from others through the other sessions. These are manual governance mechanisms imposed by the Team lead/solution architect.<br><br>All the processes make development predictable, ensures ownership as low-quality work will be seen. It is loss of face if during orchestration, your story does not work well, since you forgot something. Thus, it keeps people sharp and structures the process. Since people are sharp the chance for errors is decreased.<br><br>We can bring applications live in a day. |
| Disadvantages of interaction | Description | Architectural requirements from the workshops are inherent to the platform and do not add new features. The architectural requirements focus on robustness, security, scalability and performance. These are more quality attributes than features, thus you see changes less fast.<br><br>It is quite the organizational burden, we are spending quite a lot of time on discussing, planning and finding implementation for these questions. Thus, time spent on managing the process. For one team our approach is manageable, however I can imagine that our approach doesn't scale.<br><br>Scrum meetings are of a technical nature, this can pose a challenge if your PO is not tech-savvy.<br><br>Steep learning curve for starters / new team members, as the team is currently self-organising.<br><br>We require all stakeholders to join all ceremonies and architecture workshops, this can be demanding for your team members. | The distinction between design and technical refinements and early feedback on designs requires up-front work for relatively far in the future. The next three sprints are completely filled because of this and there is a large backlog which can fill several sprints.<br><br>Team responsible for this platform is a small island within the organisation, which might be the greatest risk. As, the organisation is relatively dependent on the team. After the migration this dependency will be less big, as this is an existing platform where other parties are connected to. Tacit knowledge of the old platform is not there currently, as the person who had this has left. However, the stability of the current team helped the team to gain an understanding of the old platform.<br><br>Team members might consider that the solution architect/team lead role has too much power. That is why he distributes architectural work, to allow the other developers to take ownership. However, there are certain things, that the architect has already seen from his predecessor, this makes it more convenient for the team if the architect does a |

| | | | |
|---|---|---|---|
| | | | design and the developers do the implementation. This way they can learn and take the design role if a similar application needs to be built. It is a bit hierarchical, but it works effectively for this process at the moment.<br><br>The fact that the architecture role lies with one person has the implication that the architect always needs to be available to explain things. Even more so, as the solution architect/team lead works 2,5 days on the project while the developers are involved full time. Therefore, the solution architect needs to be available the whole week. Not immediately, but as soon as possible, as otherwise, the team can get stuck.<br><br>Problems only arise if things come on your radar that were not on your radar. This is what happens in the extreme example of agile, where you start with development in sprint 1. Somewhere you will mess yourself up. So, I am fan of a sprint 0, where you work out design principles and architecture principles on which you can build. They don't need to be complete, as you need to adapt, which requires rework. No matter your architecture setup. The question is which choices you make in your rework or technical debt, if you choose for the short term, that is not the wisest. In our process there is understanding for this and we got the room to change the setup to make it more stable in the future. Thus, there is technical debt in applications.<br><br>A danger of a Scrum process is laziness, as it is easier to pull the whole team down than to lift the whole team up.<br><br>Due to the way the development is set up, it can be more slow and rather stiff.<br><br>Biggest risk is if a key stakeholder within the organisation or team leaves. |
| Communication in combination | Irregularly / weekly / daily / Description | Daily, weekly and two-weekly. All previously offline meetings are now held online. Offline architecture workshops. Team members are accessible and open to calls / discussions. | Daily, weekly and two-weekly. Offline architecture workshops thrice last year. Online culture that simulates offline working when necessary. |
| Impact of communication | Description | Stakeholders are aligned. No nasty surprises. Stakeholders can give insight to superiors about the situation. | Stakeholders are aligned. No nasty surprises. Trust within our team is key for our success.<br><br>Open discussions lead to architect as well as developers being able to challenges each other's ideas and convince each other. |
| Knowledge of architect / team | Description | Good. Sol. Arch. Can act as a senior dev and help more junior devs. | Devs had good base knowledge but limited product specific knowledge and improved this greatly. |

| | | | |
|---|---|---|---|
| Uncertainty addressed (types of) | Costs / schedule / quality / description | Technical (I.e., can the new platform deliver the same performance and security as the previous version).<br><br>Low uncertainty on requirements, as these are mainly the same as on the old platform. | Technical, requirement risk was not present as stories that raised questions were not integrated, and stories were generally of high quality. In other words, uncertainty on requirements was well mitigated. |
| Strategies to address uncertainty | Description | Technical: Run (performance) tests to collect data and determine next steps. No real uncertainty in requirements as these are the same as those of the old platform, however technical solutions might differ.<br><br>Uncertainty from regulation/ compliance is addressed by the PO and transformed into stories. | Requirements: stories that raise to many questions are parked or sent back to refinement. Make visible how much work new uncertain stories brought by use of research stories.<br>Technical: assign research story to backlog to investigate technical risk and solutions.<br><br>New requirements were evaluated by the team in terms of points and this score was given back to the PO. This helped in situations where the PO thought something new was less work than it was, as a higher number of points would then lead to a lower prioritisation on the backlog.<br><br>Moreover, stories with too much uncertainty were sent back for further refinement or a research story was used to address the uncertainty.<br><br>Max size of a story was 13 points, larger stories were split up. There was a preference to split up stories in general to make them more tangible, clear and achievable.<br><br>Our vision on the setup of new applications is pretty standard: we always want a micro-service application which is scalable, non-stateful, so it can pick-up on another node, lightweight and Dockered to allow easy scalability. Making use of proven technologies. This is shared with the rest of the platform, as we do not want tens of different technologies for every different micro-service. Thus, as much standardisation as possible as we are a small team. |
| Governance structure/strategies | Description | Stable team and governance structure of software development processes.<br><br>Organisation has separated teams for their services.<br><br>Technology officer that coordinates all the teams for different services. | Stable team and governance structure of software development processes. Made clear to organisation that careful development takes time and is important.<br><br>Micro-services architecture is implemented to separate primary processes from secondary processes and increase scalability.<br><br>Hard deadline for beta-test with first end-users.<br><br>Development team is flexible towards the organisation by being available and working hard to get things working. In addition, the team strives for quality, with a |

| | | | |
|---|---|---|---|
| | | | strict way of development, a lot of testing, peer-reviewing and speaking up when something is not in order or could have been done better. While allowing room to have fun.<br><br>Most governance strategies or ways of working snuck in over time, as they worked well.<br><br>Team lead/Solution architect steers towards research stories if he thinks uncertainty is large for a specific functional question in refinement.<br><br>Knowledge that was not present in the organisation has been hired externally. |
| Impact of governance structure/strategies on project | Description | Stable team and processes build trust of team itself but also of PO and external stakeholders such as operations and business. This trust is built through the ability to deliver quality at a transparent and constant pace.<br><br>There is little communication or alignment between teams.<br><br>Technology officer collects updates from all the teams regularly and inquires on status, issues and how things are going within for the service. Moreover, this role updates the teams on changes on an organisational level through the PO or implementation engineer. | Stable team builds trust of team itself but also of PO and external stakeholders such as operations and business.<br><br>Decoupled primary and secondary processes allow for primary processes to continue, while secondary are not.<br><br>Hard deadline means that the basic features should be ready by then. This requires careful prioritisation. Consequently, recent refinements were focused on prioritisation and determination what was necessary at which time.<br><br>Focus on flexibility and quality gave the team trust. They were taken seriously and seen as a partner of the organisation. This allowed the team to take co-ownership, which enables people to speak their minds and feel taken seriously. This is different from other projects where the requirements were thrown over the fence and stamped down if stories were not finished.<br><br>Research story on uncertainty on functional questions shows the impact of the new functionality that the PO considers and makes transparent the differences in estimations of story points, i.e., 1 or 2 by the PO and 5 till 8 by the team. This the PO to make better trade-offs. |
| (Changes in) management | Description | New PO, who is less tech savvy then previous PO. | Scrum master role has been abolished |
| Impact of management on project | Description | PO cannot think along on technical solution as previous PO could, but does understand that quality attributes as robustness, security etc. are important. | PO has a better eye for functional parts of the service than technical parts, however there is another, experienced role that is actively involved that has a good eye for technical parts/gaps.<br><br>Team lead/ sol. Arch. Is shadow Scrum Master, as team is mostly self-organising due to ownership of activities and speak up culture. This culture is due to the team lead/ sol. Arch. But also due to the |

| | | | |
|---|---|---|---|
| | | | context of the organisation. Shadow Scrum Master role was more at the start of the project when team members were less experienced.<br><br>Team Lead/ Sol. Arch. Seems to be a coaching mentor type of leader that nudges team members and plant ideas in them to empower them. |
| (Changes in) procedures | Description | Governance strategies determine how we align agile way of working with the vision of the service.<br><br>Developers peer review each other's code. | UX designer is responsible for designing and testing the flow of the application. Other designer has written a UI Library that can be used over different applications.<br><br>Differentiation between design and implementation phase to unburden the development team, as there is a hard deadline to go live.<br><br>People who need each other seek each other out in formal and informal sessions.<br><br>Research story on backlog to identify and assess replacements for a framework which was end-of-life support.<br><br>Documentation in JIRA, not whole documents full.<br><br>Preference for defensive development strategy: assume functions can be called that should not be callable or that certain rights could do more than they should.<br><br>CI/CD that checks code and does processes. The extensiveness of testing depends on the user story, stories concerning security, authentication or authorisation in an application then it needs to be flawless. |
| Impact of procedures on project | Description | Agile process makes sure that architecture is open to criticism and change. Current processes allow team to adapt quickly, deliver quality to the PO, show that we want to be taken seriously and how we can deliver through our own process and governance.<br><br>Peer reviews requires knowledge of different technologies and the processes under the code. | Architect picks up more difficult/experience relevant tasks but leaves room for devs to learn through smaller architectural tasks. Architect actively invests in the capabilities of his developers. Peer reviews, workshops and retrospectives are used to ensure quality of code and deliverables. Architect did the last peer review of both developers to ensure code was up to standards with the envisioned way of development and to guide the learning process.<br><br>The UI Library creates uniformity in the organisations landscape.<br><br>Distinction helps to avoid endless discussion, as for every step in the process the design is prepared in several sessions with the project manager, implementation engineer, designers and sometimes another person if needed. The team can then shoot on this design. |

| | | | |
|---|---|---|---|
| | | | Open culture creates a more viable and sustainable IT-landscape.<br><br>Research stories allowed the team to carry out this market research and present the results to stakeholders with decision making power in the organisation, for example other architecture roles. After this the decision could be build and everybody was informed.<br><br>This amount of documentation is acceptable for team and organisation.<br><br>More structured, secure way of developing. Requires more thinking on how to code. Code minimalization, no redundant duplication of code.<br><br>Automated testing, unit tests to check the code, performance testing which include functionalities of the whole platform. Solution architect cherry picks pieces of code for review, the implementation engineer tests the code functionally on the test environment. For design stories, one of the designers checks whether the flow is good. Thus, there is a layered approach. The solution architect always makes the releases, which are put on the acceptance environment by the DevOps engineer and tested again by the DevOps and implementation engineers. They also check which stories are in the releases and look at the logging for anomalies. Then it is moved to production. So, the release process is waterfall, however very streamlined and structured to ensure availability of the platform. |
| Project delivered within budget | Yes /no + description | Project ongoing | Project ongoing |
| Project delivered within time schedule | Yes /no + description | Project ongoing | Project ongoing |
| Project delivered within client quality expectations | Yes /no + description | Yes, compliments from day-one stakeholders. | Yes, various compliments each retrospective. A lot of one-time rides, little rework. Things work as they should. Thus, the perception of quality is there. |
| Recommendations | Description | If you require clearance to implement changes from your PO, do so as this can built trust.<br><br>Deliver predicably and with consistent quality to earn trust from stakeholders.<br><br>Engage your PO, business and operations stakeholders into your ceremonies: daily stand-ups, weekly refinements and biweekly retrospectives.<br><br>To ensure quality we use an automated scan that checks for issues at each change, as changes to a current system are always a risk on its own. It tests all | Put a lot of attention and care in a new person that joins the team to rebuilt trust and stability if a team member has left. Support this person, even though it will cause delays, this is a thing that you need to accept.<br><br>Changing a key person can also bring new ideas.<br><br>Disable vulnerabilities that are actively exploited. Making a mess within an application will destroy more than is dear to you. |

| | | | |
|---|---|---|---|
| | | functionalities in the technical sense.<br><br>Peer reviews ensure quality, as does consultation with the client about what to compose.<br><br>Keeping a stable team helps to build trust within team and with external stakeholders.<br><br>Use mock-ups or sketches to convey your ideas. | |

# Appendix F – Case study report 3

Table that summarises findings of case study 3.

*Table 19: Case study report 3*

| Variable name | Unit of measurement | Product owner | Enterprise/lead architect |
|---|---|---|---|
| Project description | Description | Application within a chain (of stakeholders) that needed to calculate certain things based on data from other parties.<br><br>A lot of stakeholders and dependencies on these stakeholders (their systems) in the project which made the project very political. Stakeholders could also be very heterogeneous within their own group (municipalities). Working in a chain required a lot of coordination with these stakeholders and their IT. Laws determine which data may be shared with whom. Product has a lifecycle of about 20 years. | Project is innovative as orchestration was new for this organisation.<br><br>Requirements were more political than technical due to complex governance structure and stakeholder landscape. |
| PO description | Tech savvy / non-tech savvy / description | PO views itself as the final decision maker as what is considered in the final application and what not. However, as PO's are not dedicated to a specific service, but could be re-assigned on a quarterly basis, they cannot take ownership of the formulation of a vision and capacity needed etc. | Tasked make sure that internally the customers' wishes are implemented with a team of developers. PO represents a client he barely speaks to. Moreover, the actual client has outsourced the programme to another entity. |
| PO impact on project | description | PO takes decisions that conflict with the architecture if he feels like they are withholding them. This creates tension with the architects. | The actual client or PO comes to the organisation once to explain what they need, then an internal PO takes over. Thus, there is distance between the PO and the actual client due to several delegations of power. The PO interacts with the client only occasionally. |
| End-user description | Tech savvy / non-tech savvy / description | End-users don't know their own processes and are of an administrative function. | End-user is a specific type of civilian and the administrative organisations and persons are an in-between role. Could also be the party that we do the assignment for. In reality there is no one customer, but a set of stakeholders with different, conflicting interests. |
| Role of interviewee in project | Software architect, enterprise architect, member of agile development team, | Product owner, saying no to things and stakeholder management. | Enterprise and Lead architect. Managing the architecture team and responsible for the enterprise architecture on a strategic level. Involvement in chain architecture with all stakeholders in the chain. Then transformed the Chain Start Architecture into a Procedure (traject) Start Architecture for own organisation and propagated the PSA to the environment. |
| Experience in role | < 5 years, 5-10 years, 10 – 15 years, 15 > years | 5-10 years | 10 – 15 years |
| Project status | Functional version / in operation phase / description | Functional but in a large rework phase. | Not discussed. |
| Agile framework used | SAFe etc. | Scrum with elements of SAFe. | Scrum ceremonies, sprints, refinements, demo's etc.<br><br>Cherry picking elements from SAFe. |
| (Reference) architecture framework | TOGAF, NORA etc. | Not discussed. | NORA |

| | | | |
|---|---|---|---|
| Geographical scope | International, national, regional, local, central (same building/room) | Netherlands | Netherlands |
| Agile implementation | Maturity / Description | PI's, portfoliolayer, portfoliomanager, InfraOps is the name of the operations department/teams, this and the hosting provider is where the dependencies go to.<br><br>Agile process fails to achieve product-oriented development, as the context does not allow it.<br><br>After development, the software is passed on to a separate operations team, so no DevOps.<br><br>Committed work is done in sprints of two weeks. Refinements, daily stand-ups and reviews are used. | Scrum ceremonies are done, such as stand-ups, refinements and demos. However, if this is agile, that is something that I have an opinion on. I find agility on team level irrelevant, agility in the service provision of your organisation is what I find important. By which I mean that you are able to act on changing requirements or wishes. For this there appear then some non-functional demands for the software that is realised by your teams. These are on adaptability and lead to service orientation, decoupling and containerisation, avoiding monoliths, stuff like that. Then if your client asks you to make a blue field red, you could implement that in a day. However, these are not the client requests that we get. We get a request from a ministry to build a complete service provision and two times a year, we get a load of requirement changes to implement in the next half-year release. Of course, it helps to divide this into small pieces that represent customer value, but we still collect them to release them once every half a year. Which raises the question for me, what does agile mean? We are doing the Scrum ceremonies, however if we give a demo where developers tell what has been built, there is nothing to see. As the service that we are building is an interface, there is nothing to see. The customer interacts from their own application and do not notice anything. The governance is not agile, while we try to be. We need all the specifications three months in advance, then we built for half a year and release, so nine months prior, the requirements need to be known. Where is the agility in that? I am critical, but really, I am a fan of agile and the SAFe framework, which gives a position to architecture as well. We cherry pick some elements from SAFe. However, I don't see the match between our context and the agile development process and think it is not so successful in practice. In a private context, for example with a web shop, you could implement some smart features at the frond-end of your platform and provide new functionalities directly to your partners through your platform. Providing added value. This is where I see agility, make new features today, deploy tomorrow. This differs from a more infrastructural solution to exchange data with some small pieces of functionality. |

| | | | |
|---|---|---|---|
| Role of agile in project | Description | Delivery of working code to deliver functionalities.<br><br>Agile maturity is low. | Not discussed |
| Role that represents flexibility | Role and description | Scrum master should take on this role more, as Scrum master currently holds PO responsible for this role. While the organisational context does not always allow this.<br><br>PO feels responsible for flexibility and tries to create conditions for flexibility. Also thinks that this should be represented by everybody within the organisation. | Not discussed |
| Role that represents stability | | PO feels responsible for stability and tries to create conditions for stability. | Not discussed |
| Software architecture role in project | Top-down, architect actively involved in agile development team and / or description | Not discussed, solution and enterprise architecture have been discussed. | Software architect does the design of how the software that will be built and makes technology choices. For example, they will determine how certain libraries correspond with each other and how to decouple them. A software architect goes into more details than a solution architect.<br><br>Lacking. We have solution architecture, which is positioned somewhat far from the technology, and technical people. Bridging this gap is difficult. |
| Solution architecture role in project. | | More of a pushed forward designer. Determining what the software should do. PO expects more boundaries: I.e., pitfalls, non-functional requirements, expected changes.<br><br>Architecture does not deliver working code. | You could call a solution architect an application architect, that translates the higher frames/guard rails from the enterprise architect into the functionalities of an application. However, they will not interfere with which technologies will be used to build the application. They do have an opinion on the choices that affect functional requirements of the technologies.<br><br>Explain that requirements are more political than technical. |
| Enterprise architecture role in project | | High-level design of which components there are and how they should interact.<br><br>Provides direction. High-level product vision. Some architecture boundaries. | High abstraction level on top of hierarchy.<br><br>Describe/design something that is buildable.<br><br>Setting of boundaries/guardrails what the developers can(not) do.<br><br>Explain that requirements are more political than technical.<br><br>Help to reel in assignments by discussing how own organisation could help and designing architectures to convince stakeholder that assigns the project. Ambassador of the organisation. |
| Interaction between architecture and agile | Description | Conflict between architects and developers. Developers have the final decision as they built working things. Conflict has nothing to do with agile. | The architects are the supporting troops, providing support to the teams in alignment of the software and the client needs. This does not mean that every client wish is important, it is about the goal of the organisation. Whether the teams |

| | | | |
|---|---|---|---|
| | | Architects have either too much or too little architecture on functional parts. | experience it this way, I don't think they always do. As architects are often seen as a burden, because they complicate the solution and things become more work to build. An example occurred lately, for a functionality that had to be added to an application that was some 50 lines of code. However, we architects wanted it as a service, requiring containers, namespaces etc. Making this 1,5 sprint of work. The reason being, that this functionality also can be implemented in another application. This leads to big discussions, as the steering on the agile teams is very ad hoc, what is needed now? While architecture is about what is needed in the future, could be the next two months or the next five years. If you made the wrong decisions now, it will give trouble in the future. This is something that has occurred in the past in the organisation, even though the architect had said something about it, but it was not acted upon. Another example was given, where an authorisation package could not be migrated to a new version, lagging behind multiple versions. As the package has been built in the software, while the architect advised to not do that and built it next to it. However, the decision was made to build it in the application as this saved two weeks of work. The outcome is that it took two years to complete the migration. Saying I told you so does not help in these cases. But it is frustrating.<br><br>View of this participant is that architecture lays the foundation and agile could help to design the building on top of the foundation. And both can work together to reach a good end product. If you do not lay this foundation with architecture, a team will take shortcuts that lead to issues later.<br><br>The architecture team could also work in an agile fashion.<br><br>Interaction with EA and devs is four times a year during programme increment (PI) events.<br><br>Solution architects and dev teams is more frequent, however there is no dedicated solution architect for each team. So, a sol. Arch. Works with multiple teams.<br><br>Sol. Arch and lead dev should work out technical design of business analysis process together.<br><br>Changes to the architecture are welcome, but not if they help today, but not tomorrow. |

| | | | |
|---|---|---|---|
| | | | Sometimes changes of this kind to the architecture are proposed. For this I will not alter the architecture, then I write a scenario document that shows the consequences of follow the architecture and of following the developers. This document goes to the management team. Since the management team is the one with the mandate to make this decision. But usually, this is not necessary, and it is possible to allow a temporary alteration to the architecture and to keep the architecture standing. But I would rather have had a good conversation. This is a consequence of not having a software architect, in the end the solution architecture is a bit high-level. That several technical solutions fit this solution architecture, does not mean that everything is conform the architecture. However, there is some room for interpretation in this high-level architecture which lead to problems, as there are interpretations that this architecture leads to certain solutions. But this solution is not specified enough. Writing more granular architectures lead to discussions with the teams which take up a lot of time, which lead to us not doing it. We need to find the right balance in this.<br><br>Both architects and developers have a duty to inquire and bring knowledge, ideas, interests and the vision from the other party. The solution architect should not have to do the whole technical design alone. The lead developer and solution architect should both take ownership. This ownership should be more on the side of the solution architect in the beginning and transfer to the lead developer more and more as the process continues. At some point the team lead should have most of the ownership. |
| Advantages of interaction | Description | Functional software is delivered. | A transition of ownership from the solution architect to the lead developer would reduce endless discussions.<br><br>I do see the added value of agile in good communication, refinement processes, multidisciplinary perspective on issues and division of large tasks into small pieces. These are elements that could help us.<br><br>Envisioned scenario of co-ownership should lead to a happier customer, more pleasant interaction between architects and teams, better products and less struggle. |
| Disadvantages of interaction | Description | Technical debt, rework, software requires a year to implement new/altering requirements.<br><br>Software that is delivered is more of a Proof Of Concept than a | Software that is delivered is actually a MVP.<br><br>Sol arch and tech lead working together takes time. However, this time is also attention that is |

| | | | |
|---|---|---|---|
| | | Minimum Viable Product. It is not viable.<br><br>Conflict arises, even though both architects and developers mean the same thing.<br><br>Both developers and architects cannot do their jobs properly due to the conflicts. | required to work out a technical design together. However, it is currently a capacity issue.<br><br>Due to agile way of working, especially division of work into small chunks, there is a risk, and this has unfolded, that you lose oversight of the bigger picture and create shortcomings in your fundaments. Especially if you do not give your architects enough time and mandate.<br><br>Match between agile and our context is not fit.<br><br>Large amount of rework arises later in software that is released and technical debt as the architecture is not adhered to. See procedures cell for role of governance.<br><br>Operations team need to know a lot of libraries, as self-organising teams decide for themselves which libraries they use to build functionalities in software. However, this would be better if it was done by a software architect.<br><br>Too much room for interpretation and implicit assumptions of architects in the architectures.<br><br>If we practice architecture from an ivory tower, we know we will have endless discussions. |
| Communication in combination | Irregularly / weekly / daily / Description | Nasty, unconstructive remarks have been made in the past. | Combination/interaction of architects and agile devs requires attention, however this cannot be done as there is shortage in personnel. Some solution architects and development team members cannot find each other well in communication. |
| Impact of communication | Description | Due to the history, there is still a gap between architects and developers. | If architects and developers do not find each other it shows in the quality of the software they built. |
| Knowledge of architect / team | Description | Development team is decent. | Development team knows their business well. There is not so much knowledge on what more successful organisations do and what the clients are doing. Little knowledge of and attention to architecture. |
| Uncertainty addressed (types of) | Costs / schedule / quality / description | Technical due to new functionalities: orchestration and micro-services.<br><br>Requirements. | Requirements were formulated by people that might not be able to fathom what is really going on.<br><br>Operational and security (non-functional) requirements are not always on paper.<br><br>Technical debt and legacy systems. |
| Strategies to address uncertainty | Description | No clear agreements were made with third parties, which increased the uncertainty.<br><br>Communication with other teams on what they need, how that works and what the impact is.<br><br>Do up-front design and thinking. | Developers start asking questions on requirements and started a project where stakeholders from operations and lead dev are involved in requirement identification and they like this.<br><br>Idea to determine acceptance criteria and define a process to tests |

| | | | |
|---|---|---|---|
| | | | whether these requirements are satisfied. |
| Governance structure | Yes /no + description | Organisation is steered by assignments that come from government entities outside the organisation, however this starts to change.<br><br>The assignment is then assigned to a PO and a development team. Project leader role has been changed to agile delivery role. This role is managing the external stakeholders together with relationship management to ease the burden of PO.<br><br>Teams can be reallocated every quarter, which can increase development times, as the reallocation is not based on relevant experience. | Ministry delegated the programme to a project bureau.<br><br>Legal text is basis for the service/product that is developed. However, the process from an idea to legal text takes about 10 years.<br><br>Preliminary investigation is carried out by a party that has more interest in political aspects than feasibility aspects.<br><br>This assignment falls under a different governance frame than the organisation is used to.<br><br>Organisation falls under two other organisations which leads to two types of management, or two captains on the same ship. This affects architecture, BPM, operations etc. as they have opposing interests.<br><br>Agile was used to deliver an MVP instead of the functional software product that was needed. However, this is a political use of agile, not what it was intended for.<br><br>Line of responsibility of reporting is in line with hierarchy of architecture team. |
| Impact of governance structure on project | Description | The process of winning an assignment and building it are waterfall. Solution architect already has made a design before the project reaches a development team.<br>Coordination of who will deliver what has already been arranged with stakeholders. Discussion of PO with external stakeholders is only on the details. Project has been running for three years before a PO starts on the project.<br><br>As other organisations in the chain are pressing on new assignments, there is no time to go from a functional product to a product that works well under 'the hood' and is manageable for operations.<br><br>Projects can take longer as people might be assigned to a project for which they lack the required experience, even though the experience is available inhouse. However, this starts to change as the PO's organised themselves to gain attention for this issue. Moreover, the quarterly assignment system reduces the stability of the teams and the ownerships of teams and PO's of the products that they develop. | PO, developers and architects are far away from client.<br><br>First problem statement is drafted by legal persons, not architects or developers. Feasibility is not considered in writing the legal text. Process of creation of legal text does not match the agile context.<br><br>The preliminary investigation further strains feasibility of solution. Each step described above creates more distances to factors that play a role in reality.<br><br>The different frame led to more discussion, delays, new agreements, back to the drawing table.<br><br>Due to the different captains on the same ship, devs can start building something, but it is very difficult to know whether they built the right thing and stakeholders in the chain and own organisation are satisfied.<br><br>See cells on delivery within time and budget.<br><br>Software architect shows that their software architecture is conform the with the solution architecture. The solution architect shows that their solution architecture is conform the enterprise architecture and is conform with the enterprise security frames. The enterprise |

| | | | |
|---|---|---|---|
| | | | architecture shows that their enterprise architecture is conform the chain architecture. The chain architect shows that their chain architecture is conform the NORA. Deviation is possible, however then you need to be able to account for why and accept the consequences. The acceptation of the consequences is a management team decision in my view. If it's small, it could be handled by the PO. But the mandate is currently on a high-level, if this was not the case it could be decided by another party than the management team. So, it is not the PO's decision to save time and address the compliance issues to the architecture later. |
| Management | Yes /no + description | Management did not define clear roles and responsibilities of architects, PO's and developers. Therefore, they also do not enforce these role and responsibilities.<br><br>Decisions on capacity allocation and new assignments are top-down.<br><br>Management feels the need to tighten controls due to developers going out of their way (overstepping their mandate). | Steer on working from assignment (new law implementation) to assignment, while technical debt needs to be addressed.<br><br>Managers priorities shifted over the decades from interest in the whole chain and good working ICT-solutions to risk management and managing their self-interests.<br><br>It is difficult to get your interests voiced through all the different layers of different stakeholders that are involved.<br><br>Trying to expand the architecture team to bridge the gap between solution architects and technical people. However, this is a struggle as it costs money, thus the added value needs to be clear for management. |
| Impact of management on project | Description | Lack of clarity on roles and responsibilities might be a source of the conflict between architects and developers.<br><br>PO feels like he lacks consistent capacity and mandate to do his job.<br><br>Other teams and PO's lose mandate, capacity and trust to do their job. | No/less time to address technical debt. Difficult to work agile in an environment that is assignment driven.<br><br>Leads to discussions on how much infra rework/innovation can be considered in a project which in turns limit the ability to achieve agility.<br><br>Change is slow, but possible.<br><br>Expansion of architecture team does not really work, as people leave. There is currently, one principal developer with software architecture knowledge, however this is too little. There should be a software architect in each team, who together are responsible for a software architecture across all teams. However, currently, due to a shortage of architecture personnel, it stops at a non-technical solution architecture. |
| Procedures | Yes /no + description | Development teams are not considered in the performance test that is done when the assignment is prepared.<br><br>All dependencies end up in the supporting teams. | There is no clear responsibility defined when a discussion between an architect and development team arises. The teams are self-organising and workers independent. However, they do have limited mandate. |

| | | | |
|---|---|---|---|
| | | Only sometimes does operations attend daily's or refinement sessions. There is not a lot of input.<br><br>Starting experimenting with DevOps by assigning somebody from operations to each development team. | |
| Impact of procedures on project | Description | This creates conflict where the organisation would want commitment, as the teams are not considered in the test if the assignment can be done, and they are given an assignment with a beginning and an end-date.<br><br>DevOps could help to reduce these dependencies to one point.<br><br>More stakeholders attending the refinements and daily's might help to identify problems earlier than release.<br><br>Benefits from this DevOps experiment is limited to the person you get assigned. As some operations members don't know how to setup a development environment. | The architects have no power to block a release that does not conform to the architecture, even though they would like to. This would give incentive to adhere the architecture. Now the architects try to tempt the teams in advance, by helping the teams. However, the organisation suffers from software that is released, which gives large amount of works to rework or alter. It creates a lot of technical debt. The management thinks that teams and architects should find the answers in a meeting. But in practice this often does not happen. In an agile environment, a lot of responsibilities need to be put with the team, however then the team needs to be told that they are responsible to follow the architecture. If they don't, they just do something. For example, there are multiple teams which choose their own libraries to create pieces of software. Thus, it happens that the same functionality is held in production using two or three different libraries. Which I think is inconvenient. But self-organising teams choose these things themselves. This creates problems upon transfer to the operations team, which is in itself inconvenient. As it is better to keep software within the team to ensure that they built it maintainable. Thus, the operations team has issues sometimes as they need to know a lot of libraries. |
| Project delivered within budget | Yes /no + description | No, it was over budget. | I don't have a clear view on that, I think it was reasonably within budget. There was a difference in the initial development budget. The project falls within this due to the Minimum Viable Trick and since there is extra budget for replacements if people fall sick, however we cannot always find them. But if you put all the extra work and continuous releases in the operation phase, which is normal, but are you within budget then? |
| Project delivered within time schedule | Yes /no + description | Yes, I got compliments that we delivered the first product within the organization on time. However, an extra team was needed for four months to get it in production after testing. We were the first party that could go live in the chain. | We undressed the project and made an MVP of it. The MVP has been delivered, however if you look what has been built and what was necessary, then there is a gap. The MVP has been delivered within time; however, its properties were in a minimum way usable for the customer. |
| Project delivered within client quality expectations | Yes /no + description | Client was very satisfied. However, we were late with some follow-up changes. | There were functionalities missing in the MVP, that we had to build after delivery. The client is |

| | | | |
|---|---|---|---|
| | | That is why I had to do a refactoring iteration. Almost no malfunctions. | reasonably satisfied, it performs well, gives the right answers. |
| Recommendations | Description | | There should be a software architect within each development team. who together are responsible for a software architecture across all teams. |

# Appendix G – Case study report 4

Table that summarises findings of case study 4.

*Table 20: Case study report 4*

| Variable name | Unit of measurement | Enterprise Architect |
|---|---|---|
| Project description | Description | This development process considered the improvement of interaction and the modernisation of services. |
| PO description | Tech savvy / non-tech savvy / description | PO's are usually from IT, business owners are from the business (BO) and provide knowledge to development team. However, they should do so on top of their normal activities.<br><br>In this project multiple PO's are involved as there are multiple components. |
| PO impact on project | description | Project has a PO, BO (one or two) and a tactical strategic programme manager. |
| End-user description | Tech savvy / non-tech savvy / description | Internal stakeholders as the project was a modernisation of services. |
| Role in project | Software architect, enterprise architect, member of agile development team, | Enterprise Architect, this is a separate role from the IT-lead architect. Which in SAFe are combined into one role. This is a governance strategy based on priority, availability and specialisation of the team. |
| Experience in role | < 5 years, 5-10 years, 10 – 15 years, 15 > years | 10 – 15 years |
| Project status | Functional version / in operation phase / description | Ongoing |
| Agile framework used | SAFe etc. | SAFe, difficulty to get dynamics in teams. |
| (Reference) architecture framework | TOGAF, NORA etc. | TOGAF and NORA concepts are used in a non-rigid manner. |
| Geographical scope | International, national, regional, local, central (same building/room) | Not discussed. |
| Agile implementation | Maturity / Description | Features are determined by architect and business and they enter agile process through the backlog. Epics are prioritised during PI's and allocated to development teams. |
| Role that represents flexibility | Role and Description | Enterprise, IT-lead and solution architect, they steer on making choices for the MVP, to seek out customer wishes and which types of choices to make. Also where solutions and flexibility is needed. Business act more in this regard. |
| Role that represents stability | Role and Description | IV-firm. Making maintenance plans for technology based on monitoring reports. |
| Software architecture implementation | Top-down, architect actively involved in agile development team and / or description | Not discussed. |
| Solution architect role in project | Description | Work out the general outlines of the solution. Draft a mini-business case with business, delivery of a high level feature map and global MVP definition of an epic. Safeguard the epic during development.<br><br>Give input for maintenance plans drafted by IV. |

| | | |
|---|---|---|
| Enterprise architect architecture role in project | Description | Transform ideas of business owners into epic hypothesis by determining the core of the changes and frames, the guardrails/frame where it should operate in and the added value it should deliver through a few advisory conversations. The aim is to describe the hypothesis very functionally. A template that is like those from SAFe is used. Identify hick-ups, sizing issues, external dependencies or deviations from strategy in these ideas and reject idea if necessary.<br><br>Internally, ensure that business processes are known in order to determine whether to do them internally or to outsource them. Enable coordination and choice in this regard. Identifying shortcomings and bottlenecks in our business processes and determining which services do we want to change together with IT lead architect. While coping with scarceness of resources.<br><br>Manage external impulses form chain partners, suppliers or legislation.<br><br>Proposals for big changes come from both Enterprise and IT lead architect and are proposed to the chain architecture or a concern architecture board. |
| IT lead architect | Description | Oversee the technology supply that the IV-organisation has. Which shortcomings are there in the technology supply? What should be developed, what should be bought? Draft proposals for this to business with Enterprise architect. |
| Interaction between architecture and agile | Description | A development process usually starts at with an initiative of the business. This is the beginning of a ‘change’ on what needs to known and why. This idea needs to have an epic owner which is a business owner that wants something (SAFe). This business owner can go to the portfolio managers and architects with their idea. Architects and business interact early, when there are no requirements yet, just an idea. The architects help to make epic hypotheses of the idea and try to identify the effects on the frames, guardrails and what it should deliver. This usually happens in a few conversations between the business owner and architect. The business owners needs to gather management support (investments) for their idea. Thus, usually the idea that reaches the portfolio has MT mandate and is supported on paper, with powerpoints, scenario’s, figures etc. Architect works together with business owner to draw an epic |

| | | |
|---|---|---|
| | | hypothesis. Business needs to pitch the idea in a (tactical) prioritising consultation. This is the first iteration, the idea is now prioritised and parked.<br><br>Prioritised epics are prepared by preparation team of designers and architects, they will work out detailed designs of epics in the current programme increment that need to be realised in the next programme increment based on capacity and urgency. The outcome is a rough architecture sketch of what the idea is and how the landscape should be altered, a sort of solution outline. A business case with the expected costs and benefits is added and delivered alongside the solution outline and definition of an MVP. Specialists are also involved in this process. These are discussed in another prioritising consultation by the business owners, based on the expected added value, risks, business goals and if they fit with the current guardrails etc. If it has passed, features will be defined. It is passed on to a requirements team that specifies the features with the businessowner and experts. They write featured documents which end up on the backlog of the development teams.<br><br>Allocation of work for solution outline/crude architecture sketch, which is then transformed into several feature documents. These documents then enter the agile process through the backlog. Then they are allocated in PI's based on availability, priority and specialisation. However, the 'flow' from idea to solution is more waterfall with an agile process at the end when the solution outline and feature documents have been determined.<br><br>Some themes are strongly embraced in the business side and worked out in small change teams with project managers until it is only an IT thing and thrown towards the SAFe train. Forgoing the model of doing things together. In other themes it is difficult to find business owners to embrace a theme formulated by architects based on process or technology similarities.<br><br>Architect that worked on the epic in the early phases of the idea is tasked with guarding that epic.<br><br>Architecture has conversations about changes and impediments with business. |

| | | |
|---|---|---|
| | | Deviations of the architecture can be made, but need to 'cleaned' after several months so that it becomes an unattended and monitored operation.<br><br>There is one global release train with eight to twelve teams which are very oriented on the IT solution. Since the IT solutions are relatively complex there is a low throughput/flow between the teams in the past. Some have been small islands in the past, this is something that we try to get more on the move by trying to connect teams more to flow and change. |
| Advantages of interaction | Description | Development team can come up with their own alternative solutions if they fall within the guardrails of Enterprise / Lead architect, otherwise it will end up with the architecture board or a board responsible for the chain architecture.<br><br>Stories help for development teams to clarify requirements for their services.<br><br>Direction, requests for epics or quick wins, that create more work in the long term, that do not fit the architecture can be rejected. |
| Disadvantages of interaction | Description | Technical debt / legacy systems.<br><br>Little budget for modernisation, implementation of laws and changes due to continuous rework and refactoring.<br><br>Projects that require a lot of new enablers are prioritised as low.<br><br>Political commitments can force the organisation to specific solutions.<br><br>Business owners are not capable to make a companywide aggregation and prioritisation. The business owners are steered by their managers and the management team. Thus, everything that was given management team support in the first step of the process is important. However, this is unsustainable as there is scarcity in the capacity that develop the actual solutions. Direction level mandate is needed to override this process. |
| Communication in combination | Irregularly / weekly / daily / Description | Communication seems to be formalised according to procedures. |
| Impact of communication | Description | Communication follows a waterfall like flow, similar to the process of an idea to software. |
| Knowledge of architect / team | Description | Not discussed. |
| Uncertainty addressed (types of) | Costs / schedule / quality / description | The risks and benefits of an idea are defined by the business owners with help of the architect. |
| Impact of uncertainty on project. | Description | The risks influence the decision to go forward with the idea. |

| | | |
|---|---|---|
| Strategies to address uncertainty | Description | The business owner can then pitches their idea, it's benefits and risks to in a prioritisation meeting with a group of stakeholders that cover the whole business area. A priority is assigned to the idea and it is parked. This is the first iteration. |
| Governance structure/strategy | Description | Actual business value delivery depends on co-creation by two organisations as a result of a split some years ago. This transition introduces some governance complexity.<br><br>Role of Enterprise and IT-lead architect are separated. This is because the IV-firm and business are also separated in the mother organisation. A separate IT department makes the IT solutions for other departments. There is also a separate entity with its own managing board that builds the information provisions for this department. There are also departments for general IT services, such as interaction, data processing, data and analytics etc.<br><br>There is also the possibility to offer specialised services or applications to other departments, these are prioritised in a portfolio. To prevent endless negotiations on the details, the offering party determines the realisation of the service after identification of the organisational services and applications.<br><br>Business case is made by team of IV-firm, specialists and business owners.<br><br>Solution outline is made by architects and functional designers.<br><br>Meeting culture.<br><br>Organisation has to respond to political whims. Organisation is tasked with execution and is steered by political processes.<br><br>Appointed a member of the management to take the lead in the portfolio prioritisation process to align it more with SAFe.<br><br>Stories have been formulated that clarify the strategy and themes of the organisation. |
| Impact of governance structure on project | Description | Due to the split of this organisation (A) and the mother organisation, the mother organisation (B) does not have to develop special cases for this organisation anymore. Thus these cases are pushed back to this organisation (A) if they require too much effort, even though they would cost the other organisation |

| | | |
|---|---|---|
| | | (A) more effort to develop than they would organisation (B). There is an escalation procedure for cases like these to ask for extra budget and staffing. Architects are looking at other options than the mother organisation and business starts to experiment with external suppliers or supply.<br><br>Architect represents the business and represents the IT (landscape) and trade-offs as well for a large degree. The technical architecture, choices and responsibilities on how to fill in the application architecture are with the IT-lead architect.<br><br>The acquisition of special services is a black box approach for the customer. The offering party asks for the non-functionals and interface specifications and builds with those. This creates vulnerable dependencies to other organisations when issues arise, if the offering organisation finds your case to exotic and too much work it might not be handled.<br><br>Difficult to plan working sessions to work on a theme with other party (IT/business), especially as people need to do so on top of their normal activities.<br><br>Uncertainty and political whims lead to inefficiency as plans are initiated on signals about/from political arena, but might turn out to be not necessary. People get tired of this and stop anticipating, is it worth the effort? Could I not better spend my times on things that are more certain? Or people anticipate on other things as there are more urgent signals from the political arena.<br><br>Bad governance structure on PI planning events is broken up.<br><br>People know the domain and organisational strategy. |
| Management | Description | A development process usually starts at with an initiative of the business. This is the beginning of a ‘change’ on what needs to known and why. This idea needs to have an epic owner which is a business owner that wants something (SAFe). This business owner can go to the portfolio managers and architects with their idea. The business owner needs to gather management support (investments) for their idea.<br><br>Business owners are controlled by their department management, but do not have the mandate to prioritise their own epic over that |

| | | |
|---|---|---|
| | | of others, as this mandate is on the management board level. |
| Impact of management on project | Description | Thus, usually the idea that reaches the portfolio has MT mandate and is supported on paper, with powerpoints, scenario's, figures etc. And ask the portfolio managers and architects for help.<br><br>Since the businessowners are controlled by their department management they cannot prioritise and aggregate epic's on an enterprise level, but strive to get their own epic's prioritised. Business owners occupy strategic behaviour to get their own epic's prioritised, I.e. by making several smaller epic's. |
| (Changes in) procedures | Description | Old process to come from a business idea is replaced by a new process. The former idea was to put business owners in the same room and let them prioritise the different epic's. However, business owners are not able to prioritise and aggregate epic's on an enterprise level. Now a person has been given the mandate to take the lead in this portfolio prioritising process which is in line with SAFe, in a uniform, traceable and repeatable manner.<br><br>Trying to link teams more to value streams/flows or changes they implement. As there was relatively static structure of teams, this was because the landscape was viewed as complex.<br><br>Changes in architecture that implicate external business partners require agreement from the relevant concern architecture board.<br><br>Tenders are required to do business with external partners.<br><br>Maintenance portfolio is managed with a checklist that measures the business and technical value of applications, which is approved by IV-firm and business. |
| Impact of procedures on project | Description | Business tries to throw their ideas to SAFe trains instead of developing ideas together as business and IT.<br><br>The quite static structure of the teams over time resulted in islands, there is now more movement between them. There is movement towards the Spofity model where there is steering on how much work team is expected for each team, reallocating teams if necessary.<br><br>Enterprise and IT lead architects should get approval from external architecture concern boards with chain partners. |

| | | |
|---|---|---|
| | | Tenders and integration tests require 1,5-2 years, causing delays. Pilots with suppliers are difficult as this gives preliminary insight for the tender process.<br><br>Checklist for maintenance portfolio helps to determine which elements need investments and which elements need to be phased out. |
| Project delivered within budget | Yes /no + description | Not discussed |
| Project delivered within time schedule | Yes /no + description | Not discussed |
| Project delivered within client quality expectations | Yes /no + description | Not discussed |

# Appendix H – Case study report 5

Table that summarises findings of case study 5.

*Table 21: Case study report 5*

| Variable name | Unit of measurement | Lead Enterprise Architect | Domain architect |
|---|---|---|---|
| Project description | Description | Organisation benefits form the network, on the other hand they need to comply and participate through European law. There is no harmonisation of European laws within the network, which adds complexity.<br><br>Due to the relatively high degree of automatization in Dutch processes, the reference application was of limited usability. While on the other hand connecting with much smaller organisations poses security issues. | A system that needs to exchange information and data in a network of different organisations on a European Level. The system came forth from European legislation. This requires building a piece of software that connects the own administration of the organisation to the network. There were many organisations from different countries that had to connect to the network, which made the requirements procedure complex.<br><br>There was a reference application developed which was later discontinued. The reference application was not suited as it did not allow for processing large volumes of data. So, some alterations needed to be made.<br><br>Project lasted over 10 years; majority of this time was to reach agreements on standards for the network.<br><br>As some countries had a lot of institutions that needed to connect to the network, the requirements procedure took a long time and was difficult. The result is 'a sheep with 25 legs' that you must connect to. |
| PO description | Tech savvy / non-tech savvy / description | PO acted more as a program manager. The PO might have multiple PO's under him/her, so is more of a Chief product owner. Within the program there were multiple teams that developed.<br><br>The business stakeholder, the senior user was the director of the managing board.<br><br>Drive to execute the functionalities, ensure that IT delivers on time and correctly. Tensions have developed here. | PO was from business and had multiple teams. Under this PO worked several teams. |
| PO impact on project | description | Business owners didn't care much for the capacity issues on skilful resources such as Java programmers and quality of the reference application that has been developed externally. They just wanted the stuff to be implemented and to work as planned. Technology can be a limiting factor, but uncertainty is difficult for them to cope with.<br><br>Iteratively replanning is sometimes difficult as there is a deadline that needs to be met. | Lot of communication happens through PO's, as teams listen to their PO's. |
| End-user description | Tech savvy / non-tech savvy / description | End-user is of secondary education level and needs to be trained to use the new software. This was done by the business | Staff member within the organisation which handles customer requests and raises questions to external parties or |

| | | | |
|---|---|---|---|
| | | through a train-the-trainer model. IT was more concerned with development of the software, security and performance were big topics. | answers questions from external partners. |
| Role in project | Description | Participating in the steeringroup and guarding things from outside. Take decisions on big topics. | Very diverse. Great difference in the level of detail, sometimes engaged with high-level solution design, then you hear that an operator wants to open the Firewall ports. Engaged on a conceptual level and deep technical level. Which puts you in a split.<br><br>On one hand legislation that you need to follow on the other hand design of the solution.<br><br>As architect you are often some sort of problem owner, where you regularly are playing chess with the overall PO. |
| Experience in role | < 5 years, 5-10 years, 10 – 15 years, 15 > years | 15 – 20 years | 15 – 20 years |
| Project status | Functional version / in operation phase / description | Not discussed. | Ongoing for 10 – 15 years. Most of this time was used for the setting of standards of the network at the start of the procedure (traject). The software connection is in production, however there are still teams doing aftercare and removing technical debt, I.e., making the application conform the PSA, removing workarounds with robotics. |
| Agile framework used | SAFe etc. | Not discussed | Lightweight version of LeSS was implemented by project manager that organised a common backlog and prioritisation for the project.<br><br>SAFe did not exist for a large part of the project.<br><br>Currently there is an agile roadmap being implemented to become more in line with the SAFe framework. However, this was not yet happening at the time that the software was in development. This project was leading the way in terms of agile implementation. |
| (Reference) architecture framework | TOGAF, NORA etc. | TOGAF | Not discussed |
| Geographical scope | International, national, regional, local, central (same building/room) | Central inhouse development | Central inhouse development |
| Agile implementation | Maturity / Description | Relatively outdated way of IT-development that needs to be addressed. Functionalities are determined ad hoc and solutions are already considered in this stage. While we should look at what the capability is that should be delivered to external clients or partners. Which processes support these capabilities and which processes should be automated? Do I have a functional component already? Do I have the supporting infrastructure? And work downwards like this. This flow is not present yet, only from the creation of the functionalities on. | This project took place in a situation where the organisation was working with agile teams, however there were no agreements on how to manage these teams. This is not a problem with a small project.<br><br>3 week long sprints were used for agile teams. There were also teams involved that did not work agile that had to be coordinated with for the release. Non agile teams worked with a Kanban-like system. This was on a monthly basis. Peer reviews, refinements, demos and retrospectives were used; however, the quality of the peer reviews was not up to standards. End-users were |

| | | | |
|---|---|---|---|
| | | | involved in the demos and retrospectives. However due to the number of teams involved (five) it was not possible for the architect to follow all these ceremonies. |
| Role of agile in project | Description | Not discussed | Prioritisation. The outline or frames are there, but the agile teams need to do the prioritisation and create right heartbeat of stories that are committed to come to final solution. Thus, refinements and going the to the final solution in a stepwise manner. Often using demos, to show intermediate products that show clear business value. Where I mainly look at the final solution, they mainly look I need to deliver something that already has value. |
| Role that represents flexibility | Role and Description | In agile PO and Scrum master or project manager for unexpected technical issues.<br><br>In structural cases the MVA-process is used which helps to identify roadblocks early.<br><br>Architecture is there to help with technical issues but are not responsible to tow the solution. They do look at the completeness of the solution and decomposition into building blocks and their implementation order. | At a certain point you notice that things are not feasible. Mainly teams encounter this and try to cope with this and do things different, think of alternative solutions. For example, a test tool that delivers synthetic test data. Thus, you see a lot of flexibility from the teams themselves. While the architect is busy safeguarding the eventually desired end-goal. However, the architect needs to move along with teams and provide them with enough space, as the reality is different. Thus, the architect can then allow things to occur differently for a while, while trying to move back to the solution desired by the architecture over time. Thus, this flexibility needs to be given by the architect and is asked by the business and teams. |
| Role that represents stability | Role and Description | Architecture, right decomposition and architecture principles. For example, working with the current version or a maximum back-dated level. | The chain provides stability. Here you need an overarching role. So, I took an architect, project manager, information analyst and a chain manager from the operations role to see whether there is a stable in-between situation. As we are in an in-between situation, we are working in a line and do not have DevOps teams which are responsible themselves for operations, thus while we are still in the project, this needs to move to the line. Which is not very agile. |
| Software architecture implementation | Top-down, architect actively involved in agile development team and / or description | Not discussed, but similar to application architect. | Not discussed. |
| Solution architect role in project | Description | Makes an implementation design, first high-level and then goes into discussion with the designers on how the components are going to be designed.<br><br>Needs to engage iteratively with development teams in dialogue to deliver an MVA. | When I arrived here, there were no solution architects. Currently, they are there to define frames and guidelines. Concerned with the requirements and the actual solution. Some teams do not have a solution architect yet. |
| Enterprise architect role in project and organisation | Description | Four Enterprise architects. Architecture roles are implemented according to TOGAF layers: Enterprise, business, application and technology. | Within the organisation enterprise architects do not really make a domain architecture. That is not really present. Thus, what you see is that a PSA is made, however that this scope is not well defined and elements that should be in the |

| | | | |
|---|---|---|---|
| | | Internally, keeping contact with the CIO. Keeping the architecture capability intact. Ensure good architecture within the organisation and determine architecture roles for different aspects within the organisation. Coaching and leading architects. Review and advisory work for CIO. This often means simplifying things while keeping things right concerning the more technical documents.<br><br>Externally, represent the CIO in the steering group and control external influences.<br><br>Role of architect is often to come with critical footnotes to ideas, decisions or designs. Which often makes the architect the bogeyman from the perspective of others.<br><br>Role of architect is to give direction on solutions in consultation with stakeholders. For example high-level decisions on development and further development.<br><br>Keep the stakeholders that give direction upwards satisfied and explain the conditions, difficulties. In other words giving good explanations, decision preparation. | solution architecture are in the PSA.<br><br>Enterprise architecture can be disturbing for the progress of regular projects. They need a vision or a dot on the horizon to gain insight in what is expected of them in the long term. |
| Business architect role in organisation | | (Assists in) Formalisation of the organisations processes and information flows. | Business architecture is relatively immature within the government sector. |
| Domain architect role in project | Description | Fitting the project within the complete landscape of the domain that the domain architect manages.<br><br>On one hand, defining high-level architectural building blocks of the solution and fit these into the complete architecture landscape. On the other hand, coaching the solution architect with working out the high-level design and drawing up the PSA.<br><br>For their own domain, they update final architecture, the architecture model.<br><br>Look at the execution and technicalities through the solution architect. Help with the development process, application of the right technology, clearing of roadblocks, ensure a connection between the technology and domain party that is needed for the domain.<br><br>Support solution architect in MVA process. | Overall architecture. Sometimes acting as panacea (oliemannetje) as there was a lot of discussion among various 'blood types.' For example, mainframe developers differ greatly from Java developers.<br><br>Formal responsibility is to give advice on what the solution should adhere to and to safeguard that things are going the right way. Escalate if things deviate.<br><br>Tasks are broader. On one hand sketching the solution, while on the other hand thinking along whether the solution works and if the solution fits in practice and adapt the solution.<br><br>So, there is a gap between the formal responsibility and tasks that one is doing.<br><br>Being available to explain the solution.<br><br>Help teams to overcome issues. |
| Technical architect role in project | Description | Need a connection between a technical architect and domain architect or solution architect. Need to look at completion of the picture or ensure enabling | Not discussed. |

| | | | |
|---|---|---|---|
| | | features, such as network connections, server definitions, middleware components.<br><br>Technology architects ensure general development building blocks. They are involved in how software is developed, the facilities with which software is developed. | |
| Interaction between architecture and agile | Description | Working under architecture is held in high esteem within the organisation. However, development teams can deviate from the architecture under circumstances, I.e., high business pressure, but get additional requirements to steer back to working under architecture.<br><br>Criteria are defined surrounding the enablers in the PS;, however, the PSA is more to set frames. To do the actual infrastructure order with an infrastructure supplier some high-level design engineering is needed. Capacity needs to be specified with the sourcing partner. This needs to be addressed up-front, in every project we see that this is an issue since the organisation has been transformed towards agile. So, the conditions that are needed to deliver software are an issue.<br><br>This was not used in this project: iterations of the Minimum Viable Architecture. First iteration is a high-level sketch of the architecture. This works well if a package is partly or completely implemented. Then take a look at how well the requirements are defined and apprehend them into the business architecture, deliverables, thus, processes, information flows and a few important architectural building blocks in terms of functionalities and check whether the composition of functionalities is good, in terms of how the market looks at them. This is done with business and technical people to determine what is possible and what is not. Since they have had information on this, they sharpen their requirements. Then we move to the second iteration where a good picture is created on how to implement something and draft a plan of requirements. This is done from an architecture perspective in combination with high-level design: solution architecture and design. The final iteration of the MVA is more like a PSA, as this deliverable is used to build, implement and configure.<br><br>In the first two iterations the architect is working with the ones propelling the changes, while in | Architecture does coordination with other parties and defines a PSA. Agile team does work in PSA.<br><br>The interaction differs at each organisation that I have worked. In some private sector organisations, there was a very layered architecture from enterprise level to solution/lead engineer who did the technical design. In another it was very different, there a solution architect designed the solution, sold it to the architecture board and helped the team to implement the solution. This included for example, working out the sequence diagrams on how things should work in the tool and used multi-disciplinary teams that were pulled through the whole 'chain.'<br><br>Domain and solution architect tried to be present in the refinements as much as possible. In addition, they tried to follow what was discussed in the demo's as much as possible to determine the position of the project and see what has been built. However, attending all the refinement sessions is not possible for one architect as they happen in sync for five teams.<br><br>The definition of Ready and definition of Done need frames from a PSA. This is where agile and architecture interact. You need to provide direction, otherwise it is not possible to let these team cooperate. This coordination is needed to determine prioritisation of functionalities in the sprints. You cannot just build a front-end for example, as you need data from another system. Thus, this service needs to be realised. This needs to happen in a coordinated way and requires coordination among teams. A PSA is a great help to determine the goal of what the teams are building. This all needs to happen in discussions, what are the steps? Who should play a part and how do they work together? How will we build a service? How will this service be used by the customer? How will you cope with things that cause asynchrony as they take longer to develop? |

| | | | |
|---|---|---|---|
| | | the second and third MVA the architect is more involved with people that will deal with the change and how it should be implemented and designed.<br><br>Enterprise architect has no contact with developers, but with senior stakeholders. Domain and solution architect do have contact with developers. Enterprise architect does interact with domain architect.<br><br>Architecture builds a high-level roadmap and defines frames for the 'whole', and for the order in which deliverables should be delivered in respect to the deadline. Within certain phases iterative development can be used.<br><br>Goal is to have solution architects that work in development teams but is not feasible for the time being. For this you need an organisation that is very mature in agile and has architecture knowledge integrated in the development teams. This goes beyond the function of solution architect and means that team members take on the role of solution architect. For this your developer community needs certain architecture skills and your organisation needs to develop towards this. | |
| Advantages of interaction | Description | Focus on a specific discipline and everybody can keep focus on their own context and know their dependencies. A lot of domain parties are dependent on developments within the technology architecture.<br><br>Ability to adjust, less wandering in the process. Due to the issue with BUFD that you find out that something is estimated wrong when it's too late, this is a big advantage.<br><br>Motivate people intrinsically to deliver things.<br><br>The learning effect of a team can be increased through more, smaller repeated steps.<br><br>Provision of frames by architecture is an enabler to provide the freedom that agile needs.<br><br>Development teams can alter from architecture, but solutions are sought to get back under architecture again together.<br><br>MVA-process helps to identify roadblocks early. | We do have an example of a solution architect that forms a bridge between the high-level PSA and can make matters concrete. For example, things that were so apparent for me as domain architect that I did not write them down. The solution architect can help here and point to the relevant standards, instead of having endless discussions on small things. Here the solution architect who knows the architecture frames can translate them to the programmers and developers. As sometimes you are blind to these things.<br><br>Dialogue instead of red card procedure (see procedures) gains more commitment from development teams. |

| | | | |
|---|---|---|---|
| | | Attracting enough architecture resources to organise frequent interaction is difficult to achieve for organisation. | |
| Disadvantages of interaction | Description | Tensions with agile as things come together in infrastructure. We will not develop a distinct system and dialect for each pillar, you want a certain amount of standardisation. Thus, you are forced in a corset of standards. In addition, you have things like systems, network connections, certificates, user id's and things outside the organisation, these need to be arranged before they can be realised. This preparation work needs to be done up-front and is the runway. The enabling features need to be considered. This is a tension as it is an architecture (from enterprise to solution level) task to manage these things. | A complete up-front architecture where everything is known does not exist. This is the definition of an ivory tower and will create an endless process of identifying and changing things. What one writes today is no longer relevant a year later. The PSA that included everything never existed. If I address all issues in the architectural designs as expected by some of my colleagues, how will it be feasible for the development team to make the deadline? Not to mention the fact that you will end up in analysis-paralysis and will never start building, as you will always encounter new things. |
| | | Criteria are defined surrounding the enablers in the PSA; however, the PSA is more to set frames. To do the actual infrastructure order with an infrastructure supplier some high-level design engineering is needed. Capacity needs to be specified with the sourcing partner. This needs to be addressed up-front, in every project we see that this is an issue since the organisation has been transformed towards agile. So, the conditions that are needed to deliver software are an issue. | No clear definition on what should be in the PSA, some make it too high level, while others almost include snippets of code. The result is either a lack in guidance or a lack in freedom for the development team. |
| | | Resources, architects want standardisation of tools, while developers want to work with the tools they know. Because of this I have never seen the learning effect achieved due to the lack of personnel. | The frames that we give are not sufficient, we need to supply standards for development, guidelines, how to approach a peer review of a colleague that sort of things. However, we are only advising a bit on this part. This is not for us as CIO-office but for IT themselves. We give some advice on the solution direction, with our knowledge and expertise as mandate. But we have no official mandate. |
| | | To achieve the added value of agile, people need to change their task distribution in teams and become more multidisciplinary. | Problems occur if a solution architect leaves, a gap falls between the senior engineer, who had a clear picture of how things tied together across teams and the team. The solution architect should play a role in this, but could not enter, as the PO of the team said we determine everything ourselves. However, they had too little oversight of what played in the chain. In the end the solution architect was able to earn their spot. |
| | | To achieve the effectiveness of agile, such as the predictability of bur-out charts and learning effects the agile, you need to have it implemented on a certain scale and for a long enough period of time. | |
| | | Old environment and stringent standards make it difficult to recruit good developers, also considering the salary that could be offered. | More projects than solution architects, we have too little people and that is a problem. A sort of ping-pong occurs, where things are discussed that make me think, that is not right, we agreed on something else. Then it appears that a solution architect is on holiday and confusion and grey noise appears. Hampering the productivity. |
| | | Misconception that agile is a warrant to develop as one sees fit. If you take agile too far, then it becomes difficult to plan. | |
| | | Tension of short-cyclical agile methodology with hard deadlines from legislation. | Since we don't use the traditional agile where teams stay responsible |

| | | | |
|---|---|---|---|
| | | Need to find balance between short term planning and working ahead but need to have the plannability of your epics in mind. So, after the epics are refined in user stories is fine, however big forces start to shift in the political arena if things are not executed on time. | for the things they built, the effects of attrition are worsened. Transfer to operations requires more documentation and pictures of the main ideas of the code, as they have not developed the code. How to approach documentation in a chain is a tension.<br><br>There are large amounts of large amount of work that is lagging behind. These are things that the business does not see and that is where the money comes from. While things are going fine, the business does not notice and that is the problem. This is an area for improvement.<br><br>Tensions between IT and business create shadow IT like robotics, which adds business value but are not meant to act as a structural solution. These robotics solutions created operational and continuity risks.<br><br>What you encounter often in architecture is that solutions are purely conceptual, a paper tiger, but reality is different. While you need to take things into production to comply with legislation and add business value. |
| Communication in combination | Irregularly / weekly / daily / Description | I don't really interact with the developers, but the domain architect in combination with the solution architect does. I mainly interact with senior stakeholders. | It was impossible to talk to an information analyst as an architect because of the fact that one was an architect.<br><br>Communication often occurred through the PO.<br><br>I started to discuss the PSA in sessions with teams. Which helped enormously to get people on the same line. However, it is difficult to get continuity in this. As the teams know a fair amount of attrition. This requires more documentation in my opinion than most agile thinkers agree on. |
| Impact of communication | Description | Enterprise interacts with domain architect, who interacts with developers and solution architect. The enterprise can then manage the senior stakeholders and communicate conditions and problems that were encountered, account for them, explain and prepare decisions. While the domain architect executes the technology through the solution architects, by helping in the development process, implementing the right technologies, clearing roadblocks and connecting domain and technology parties. | It is important to keep the distance small. However, if there are many teams, the amount of interaction moments grows and it becomes difficult to plan.<br><br>To prevent architecture from becoming an ivory tower, you need to think along with what happens in reality. Thus, requires adapting to new insights and giving in to developers' ideas. |
| Knowledge of architect / team | Description | Don't know about development team. Architects have a good distribution of skills on focus areas and experienced, versus learning. | If documentation lacks, the gap is huge if a person with a lot of knowledge leaves.<br><br>The ability of teams to oversee the whole chain is limited. We are dependent on local heroes that have |

| | | | |
|---|---|---|---|
| | | | been around longer. However, this is a vulnerable situation to be in. |
| Uncertainty addressed (types of) | Costs / schedule / quality / description | Technical, how to integrate above average automated system to a more general system. So, how to get information out of specific Dutch systems into a general European system.<br><br>Technical: quality and discontinuation of reference application.<br><br>Staffing<br><br>Requirements: specs of reference application were clear. However, were dependent on large parts the reference application as the whole application needs to be developed in-house which is a big task. | Technical: quality and discontinuation of reference application.<br><br>A lot of important technical people are aging towards pension age that are responsible for legacy systems.<br><br>Requirements: not so much uncertainty in this area as the specs were pretty clear. However how to fill these specs and organise interactions between teams was difficult. |
| Impact of uncertainty on project. | | Decision to build own application. However, this gave difficulties as well as it is difficult to attract skilful Java programmers.<br><br>Difficulties with non-functionals as reference application was not built for high level of automatization. Functionally the reference application was good. | Technical: Decision needed to be made on how to continue.<br><br>Knowledge is leaving the organisation and this will become worse.<br><br>Requirements: the reference application with limited documentation which made it less usable. |
| Strategies to address uncertainty | Description | Try to create co-creation on a new reference application instead of doing it alone as the Netherlands. | Technical: Decision was made to develop own application with other parties in Europe.<br><br>Need to gain more transparency on how the technical systems work that these people built.<br><br>Requirements: this is not where the main problems were. |
| Governance structure/strategy | Description | Hard deadline due to European regulation.<br><br>Governance structure is mirrored in architecture structure.<br><br>Political arena does not give enough time to the executing parties to get their affairs in order and test the executability of legislation.<br><br>Complex legislation due to stacking of legislation. | Decision was made to make use of the reference application. However, it was not fully suited for the organisation as it was developed for less automated organisations.<br><br>Later, it was decided to develop an application in-house instead of using the reference application. As the reference application was discontinued.<br><br>Hard deadline to be connected to the network.<br><br>Organisation needs to be more open to outside world, which is a broader issue. This requires technical changes to current systems and development of new systems, interfaces and UI.<br><br>Issues on cooperation of agile teams and how to get the right teams and do prioritisation across teams.<br><br>There are different systems with different needs in the organisation. They can be classified using the Gartner framework for example; systems for interaction are more suited for agile. While more |

| | | | |
|---|---|---|---|
| | | | infrastructural or organisation critical systems benefit from stability, less change. This needs to be considered in the architecture.<br><br>An overall backlog was implemented for the project to plan and safeguard progress.<br><br>An overarching coordination for the teams missed. There were some agile and Kanban teams, who had their product owners and they needed to work it out with each other. There was a project manager that worked according to the PSA, however the project workers listened to their product owner which prioritised their own backlog.<br><br>An agile roadmap is implemented which lead to a lot of changes in roles on the business side. However, business has a tendency to do so without finding agreement with IT and the CIO-office as they are less flexible in this regard. Leading to shadow IT solutions by the business stakeholders. |
| Impact of governance structure on project | Description | Java development teams can work agile in sprints; however, the deadline stays fixed.<br><br>Each business unit has a domain architect, which in turn works together with one or more technology and solution architects.<br><br>Creation of hard deadlines which have no attention to feasibility for executing organisations, thus architects and developers as well. This creates tension with agile development. On the other hand, it helps for developers to work with refinement, manageable chunks of work in sprints and seeing the results.<br><br>Long term program is initiated in government to renew systems, ensure the right decoupling and simplify systems. This program requires a lot of time and budget. | Thus, some changes needed to be made to increase performance, but the reference application could serve as a basis.<br><br>There were a lot of small organisations that do not have the capacity to develop their own application, this comes with security risks other organisations connected through the network. This organisation was one of the first to be connected. Things work from a business perspective, but from a technical perspective it is still quite primitive. So, there is still a bit of technical debt and some trouble with the 'technical adapter.'<br><br>Required Business Use Cases (BUC's) need to be finished on the hard deadline.<br><br>Resistance from inside the organisation from those who want to keep things as they were.<br><br>Agile roadmap was implemented to identify which teams there are, how they could cooperate and control the use of resources. However, some architects keep their own ways which are more a traditional ivory tower style than in dialogue.<br><br>The front-end allows for more room, while the back-end needs more clarity, outlines and a clearer architecture vision. I think that with architecture we do this too little, provide too little or too much hinderance through outlines, guardrails or frames. Currently the teams are disturbed by the |

| | | | |
|---|---|---|---|
| | | | exploration of new technologies by architecture. We cannot hinder the processes that add business value as architects.<br><br>Overall backlog helps to reach its goals and to deprioritise things that don't add business value.<br><br>Stakeholders who previously found each other can now no longer find the right role to engage for their project. This is a risk for current projects as it affects everything from requirements elicitation, to reaching agreement on requirements with each other and the realisation itself of the software. |
| Management | Description | The PO changed two or three times.<br><br>PO worked together with a project leader that took care of enabling factors. | Management made a decision in the past to replace legacy systems with new systems.<br><br>Due to the lack of overarching coordination, the project manager could not manage the situation. Thus, a virtual team that was able to look across disciplines in the chain was created. These overarching issues were discussed by this team, there were two variants of these meetings. One for more functional things and one for more technical things. Decisions were taken in these meetings to create overarching coordination. Additionally, this helped to discuss things from which it was not clear where to discuss them previously. However, this governance strategy was implemented only in this specific project, creating an issue for new projects. |
| Impact of management on project | Description | There were struggles with the enabling features or the so called 'runway' from SAFe. This were mainly infra issues that needed to be addressed. This was addressed more project based. | Implementation of new systems failed. Now the decision has been made to refactor the system. |
| (Changes in) procedures | Description | Changes are only discussed with developers, domain and solution architect once a decision has been made on a management level that leads to a change.<br><br>You want to allow developers with responsibility and accountability to choose their own development and development and test environment if they develop code according to certain specifications and coding norms. Which comes with the accountability of the complete development process, including back-up, no loss of work, no loss of productivity, working in a secure way. | Worked initially with big room planning, which was dependent on (in)formal hierarchy.<br><br>Since there is no real demarcation of roles, the contents of the PSA are personal tastes of the responsible architects.<br><br>Breaking up of monolith and turning them into services gains resistance from certain teams. The team members have a power position as they have knowledge of how critical technical systems work.<br><br>There is a procedure for alterations form the architecture. An architect can give off sort of red card that can be escalated up to the non-executive board. The team needs then to rework the solution so that it falls under the architecture again. |
| Impact of procedures on project | Description | Developers, domain and solution architect are not included in the | Business process owners were pushing and pulling on projects to |

| | | | |
|---|---|---|---|
| | | decision-making but do have to ensure the right execution.<br><br>This is not feasible as things could easily leak to the outside world through a developer's pc which causes big problems. This is where it stings, you cannot or do not want to leave this to people, as you do not want to be at risk. | get what they want. Was a chaotic process. Changes are being implemented by portfolio management combined with information management. Now the organisation is in an in-between form, Agifall, where agile projects exist next to combinations of agile teams, agile teams that work in their own space and large projects with project managers.<br><br>Discussions arise between architects on why something is (not) included in the PSA and solution architecture.<br><br>These are the elements where people with technical knowledge are also reaching their pension age, which worsens the risk of losing this technical knowledge before the system has been refactored.<br><br>The refactoring for the red cards stays on the backlog for ever, as nobody commits to address the work, it is not prioritised and the architects are side-lined. So, it is better to sit with the teams in a more constructive manner how to rework towards the architecture and put it on the backlog as was done in this project. |
| Project delivered within budget | Yes /no + description | All the business use cases and structured electronic document implementations have been made, however, there is some development still going on. | The system is in production and all business use cases and structured electronic documents are supported. We are completely compliant, as the business value is totally there. There are a few changes that need to be coped with and the solution is not completely conform the PSA, there are still several workarounds implemented that need to be phased out. Thus, the solution is in a state of aftercare. For this extra budget has been allocated. |
| Project delivered within time schedule | Yes /no + description | Most problems were present in the planning and making the delivery deadlines due to issues with resourcing. | See previous cell. |
| Project delivered within client quality expectations | Yes /no + description | Good. | We had a lot of issues with changing from the reference application as front-end to building our own front-end. However, this user interface is experienced as better. So, in the stakeholders were satisfied. |

# Appendix I – Case study report 6

Table that summarises the findings of case study 6.

*Table 22: Case study report 6*

| Variable name | Unit of measurement | Product manager | Domain lead |
|---|---|---|---|
| Project description | Description | Sales platform that needs to accommodate consumers, businesses. But also new financial and supply chain functionalities. This case study gives a perspective of how different teams work together on a specific solution. | Reflection on various projects around a platform that accommodates consumers as well as business clients. |
| PO description | Tech savvy / non-tech savvy / description | Business, more senior product owners are more technical. PO has a product. | From business. Interact with the requesters in the markets and businesses to align requirements and shaping their own mission and roadmap for the capability that they are responsible for.<br><br>PO accepts work, is part of the demo. |
| PO impact on project | description | Some PO's are involved in writing stories, but most PO's take a backseat when features are decomposed into stories. Asking critical questions. | Do enhancements to existing, running capabilities and transforming topics. The latter includes formulating a vision on where to go with that capability In the next 1-5 year time frame. |
| End-user description | Tech savvy / non-tech savvy / description | Interaction with end-user via online test panels platforms for qualitative data on user experience and AB tests for quantitative data on user experience. Feedback from these processes is fed into the Sprint cadence, so refinements etc. | PO represents end-users, because in many cases there are multiple end-users and requesters that need to be shaped into one vision.<br><br>In more complex cases, the end users are directly involved in demos, user acceptance tests and the deployment group. |
| Role in project | Software architect, enterprise architect, member of agile development team, | Product manager brings together Senior product owners in a functional way, rather than a hierarchical way. Coaching them to make the right choices and set the right priorities in 'putting the puzzle together' and managing dependencies for example. Helping the PO's to do planning etc. in an agile way through project management. Role and responsibilities are fluid because of the agile processes employed in bubbles. | Head of IT of Direct-to-Consumer (D2C) within organisation. Overseeing the digital portfolio related to directive business.<br><br>Various roles, currently a leadership role, which means accountability for operations, development, everything. Overseeing multiple delivery managers, solution managers and architects that are working in a team, who are distributed across many different IT scrum teams that deliver and run a D2C portfolio. Meaning responsibility for the operational phase of everything that is in production. |
| Experience in role | < 5 years, 5-10 years, 10 – 15 years, 15 > years | 5-10 years, previous experience as a developer, as well as business background. | 10 – 15 years |
| Project status | Functional version / in operation phase / description | In operation | In operation |
| Agile framework used | SAFe etc. | SAFe, with own flavour. Project management and horizontal coordination are added. | SAFe, done with implementation in department. However, this differs across departments in the organisation. Continuously changing and improving it to make it adapt to the situation of the organisation. |
| (Reference) architecture framework | TOGAF, NORA etc. | Not discussed. | Combination of layered and modular architecture. Architecture |

| | | | |
|---|---|---|---|
| | | | is not highly layered such as in TOGAF. |
| Geographical scope | International, national, regional, local, central (same building/room) | Decentral, development capacity is outsourced to another continent. | Decentral, development capacity is outsourced to another continent. |
| Agile implementation | Maturity / Description | Stories are planned two weeks ahead. Epics and enablers are determined by a quarterly handshake process. The stories use, two, two and a half sprints of up-front planning. The enablers and epics are designed as a solution. | Demand in D2C domain is very flexible and dynamic. This creates tension with the quarterly Programme Increment (PI) planning sessions that SAFe prescribes. Since the demand is not laid out a quarter, but only 2 sprints ahead, it does not make sense to do an extensive up-front planning session. As you end up planning for stuff that you don't know enough about to plan effectively, and your plans will end up being inaccurate. You end up faking the situation a bit and making guesses, while you know it is going to change. So, for programmes with extremely dynamic scope, so more continuous improvements/optimisations, which are usually small features that you deliver based on what is most needed and delivers the biggest value in a situation for an area, we still run release planning sessions on a quarterly basis but use it more to align strategic direction with the entire group. Align on key areas that will be touched, without going into details of the whole PI planning, dependency mapping etc. This happens sprint by sprint as you go |
| Role that represents flexibility | Role and Description | Refinements and retro's. But also the sprint rhythm itself. See cells on uncertainty.<br><br>For fundamental changes, there is a bi-weekly portfolio management meeting with the markets and business. Shifting timelines and changes are discussed here.<br><br>The product owner makes a call, in case of changes, if it concerns multiple product owners, then the chief product owner makes the call. | We do this for multiple levels, including AB tests continuously to validate the requirements that are coming in and to get end-user feedback from consumers on what really works. Thus, in order to find out what the best solution is and to get that in your software development life cycle.<br><br>For this you need to set up your architecture in a way that allows for this flexibility on various levels. It does not have the be altered very month, but you can use a microservice based architecture and is relatively headless decoupled architecture. So that you don't kill your flexibility with complex and end-to-end solutions. This allows you to incorporate AB-test results, late requests and other things when you need to change direction and implementation or part of the UI only. Thus, enhancing flexibility to a degree.<br><br>In development you must be flexible enough to accommodate new and last-minute requirements coming in, but the flexibility part should happen upstream because, as a developer you are likely using the same process irrespective of which requirements comes. You need to make sure it's well groomed, well understood, have acceptance criteria and solution designs. In essence, |

| | | | |
|---|---|---|---|
| | | | everything needs to be ready to be taken into Sprint, no matter if it's a new requirement or some that you've been planning for years. The roles up-front determine the functional scope as well as the big architectural setup and solution designs and are thus more critical to enable flexibility. |
| Role that represents stability | Role and Description | The bubble, combination of PO, solution architect, team leads etc. | If you look at stability of technical platforms, it is an IT responsibility that you can organise in several ways. 1) Organising separate operation IT-groups that are responsible for stability, making sure the platforms are up and running and performing according to expectations. 2) You can setup DevSecOps groups who are taking responsibility for both the Dev cycle as well as the Ops cycle. In this case the product owner is responsible for technical performance and that functional things make sense. This means not focussing on if the service is up and running, but on the user journey. Can users find what they are looking for and complete their goals? |
| IT-architecture implementation | Top-down, architect actively involved in agile development team and / or description | Multiple solution architects are involved. | 1-to-many relationship, so an architect is overlooking a larger part of the landscape. Even though there are architects who are deep experts in a particular technology, in most cases they are broad end-to-end experts in a set of technologies: T- or Pi-shaped specialists. So, architects create an end-to-end architecture instead of a technology specific architecture. |
| Solution architect role in project | Description | Solution architecture works horizontally across different technologies. | Solution architects are responsible for full stack, including infrastructure architecture, application layer, the whole thing. There are also more deep experts that take care of the infrastructure, albeit that it isn't changed that often. So, while the functional architects would have end-to-end responsibility, they do rely on deeper infrastructure experts if the infra setup needs to be modified. |
| Enterprise architect architecture role in project and organisation | Description | Enterprise architecture determines which software packages are bought and which functionalities are expected from bought or licensed software packages. Enterprise architects together with procurement negotiate and sign a contract with a supplier for a certain software package. | Not specifically discussed. |
| Interaction between architecture and agile | Description | Transitioning from waterfall to agile development. Currently in an in-between hybrid situation, as there are agile processes for development. But dependencies are managed through a waterfallish project management approach, as there are various teams and technologies involved. Thus, there is a combination | PI's, product increments and release planning sessions every roughly every quarter to determine basically what ends up on a release train. As part of that, basically we have a what we call a demand intake processor, requirements gathering process that happens with a number of product owners, tribe leads, markets and other parts of the |

| | | | |
|---|---|---|---|
| | | between waterfall project management and agile software development.<br><br>Relevant stakeholders, including PO's and solution architect are taken into a bubble, that looks horizontally across teams. This bubble designs an up-front solution and works in an agile way by gathering new insights along the way. Each bubble determines the balance in up-front and emergent design based on their context.<br><br>Solution architect makes the first iteration of a solution and then the tech lead, product owners, senior representatives of the test team and in some cases business analysts can shoot on the solution.<br><br>After challenging the solution, the solution architect is going to help the team to create features and two or three epics. The product owner starts working out the features and the epics with the stories below it. As the individual stories do not add value, they do so when they are combined into a feature. However, a feature can be composed of 5 different stories from different technologies. This is easy for the PO, as they are more business oriented, so the feature can describe acceptance criteria, end-to-end outcome etc. This is interesting because this is too technical for a product owner, because it is in a different technology. Thus, the product owner does not go deeper than the feature level.<br><br>The stories contain acceptance criteria and test scripts already this way. The solution architect is involved in the decomposition of the features into stories, which is a more technical discussion. Therefore, the PO is only sometimes there and the solution architect, tech lead and business analyst take it a step further. The PO is still in the lead, but the solution architect reviews the written feature. The features is then documented and referenced on a wiki. The solution architect also reviews the stories. There is interaction between the tech leads and solution architect, for example on use-cases that the solution architect did not think of before. This is a very iterative process where everybody brings their own perspectives and knowledge to the table. This process is used for new features, not for features that the tech lead can handle alone. | organization, those requirements get translated into initially high-level designs. Where the architects get involved, then they are translated into lower-level solution designs where normally our what we call technical leads per team get involved and finetuning the large level architectural direction to smaller level solution design and then the solution designs are used by the actual developers to deliver the solutions according to those specifications of those designs. Then the work gets tested.<br><br>Role of architect depends on, two use cases. 1) capability that does not exist, architectural white spot. Ongoing alignment of business on strategic and tactical level to determine business and IT architecture landscape. Which solutions are in place to support the bigger capability problems and which solutions are not in place? The architect gets involved to explore how to fill that white spot, that exploration is usually a Request For Proposal (RFP) process, which is like a tender procedure, meaning that you analyse the market. Before you do this, you must make sure you understand what the business requirements are on a more global scale, then you analyse the market to find out which tools are out there to fulfil those requirements. Then you do the deep assessment to determine which tool you want to buy or develop in-house. So, the architects are involved on the first step, at the strategic tactical level to determine what solution should be chosen and how it would fit the entire ecosystem.<br>2) A capability is in place, meaning the architectural selection has been done. The tool is there. The tool is being built, deployed and enhanced. In this case the architects are involved in several stages. They are involved up-front in getting their requirements and determining if adjustments need to be made to access or integrate between systems. Or are only minor adjustments needed and will the Tech lead do? If it would do with the tech lead, then they create an architectural design according to that and are still involved in the actual execution of the project. To ensure that what is delivered ends up filling the architectural direction. Lastly, the architects are involved in bigger cases of production problem management. If there is a structural issue with a capability. It must be bigger than just a few patches here and there, in these cases architects are also involved to look more structurally if the direction needs to be changed or a different solution would solve the problem. This is |

| | | | |
|---|---|---|---|
| | | The solution architect has a lot of touch points in the refinements.<br><br>Department is going for a headless architecture setup; this means that the front-end back-end are decoupled. So, front-ends can be plugged into a 'socket' and gets the right answers. | done in a very agile fashion, so this all happens very close in time and in an iterative way. In use case 1 this means that if demand intake is today, then execution happens next year and you see it in production a year later. In use case 2 a demand intake today, gets planned into a sprint one or two sprints from now and then it's in production immediately and after incidents start popping up these are to be addressed. The architect interacts with the development teams in these cases through the team/technical leads and product owners. This allows for alterations to be made if the architecture design does not fit for the development team. |
| Advantages of interaction | Description | Even though the sessions where the first iteration of a solution is discussed can be heated, it adds a lot of value since you can make a discussion to do some dirty work if there is a lot of uncertainty to throw the MVP away and learn from it. See whether it moves the needle on the business KPI's. On the other hand, if it is known beforehand that the situation is complex, or it has been done a few times and is known to bring value. Then a bigger development investment can be made, which takes longer, but creates a fundament that can be expanded upon.<br><br>The headless setup enables agility for development teams.<br><br>The skill set of the teams is very technology driven. The teams do not employ multiple technologies. Therefore, the teams and tech leads are unable to look over the wall. The solution architect can cut across these teams and technologies and act as a bridge. Knowing a lot of technologies, but not the details. This can prevent issues occurring downstream in different parts of the solution that the tech lead cannot foresee. Additionally, this difference in perspective and level of details adds new insights to the discussions. | Allowing architects to formulate a vision as on how to not only solve current product owner needs, but also identify what are the future capabilities that we might have. Therefore, up-front find out how to fill white spots or create white spots in the first place.<br><br>The architect would be able to formulate a solution that will avoid having to do rework later. If you align requirements up-front and understand deeper what is really needed, you will end up building a solution that satisfied all the different needs instead of building a sort of step-by-step solution that you may need to modify or rework entirely. For example, your solution works functionally, but performance wise it is killing your system. |
| Disadvantages of interaction | Description | You cannot fit a big platform to an agile setup because the teams would become too big. As you would need to fill the team with different technologies, which would create teams that are too big, as around 9 people in a team would be ideal for a Scrum team. But you would end up with teams of 25-30 people, which is not doable.<br><br>To add to the previous bottleneck, you cannot find T-shaped people, that know all these technologies and the associated processes, such as building services. Thus, | Bandwidth. Everybody is busy with a lot of activities. Some daily, some operational, some tactical, others more strategic. So, it's a requirement for architects or other more senior positions to properly balance that. Thus there is a risk that if a person does not manage their time and overall role well, they might end up going to far in one direction, having no more time the others.<br><br>If bandwidth is not managed well or an architect becomes unavailable, then multiple teams can suffer from |

| | | | |
|---|---|---|---|
| | | proper project management is needed to manage dependencies that the multiplicity of technologies creates.<br><br>Since the solution architect makes the first iteration alone, the solution design depends on the solution architect. Interviewee describes how with one you need to steer on more thoroughness, while with the other you need to steer on more speed.<br><br>The meeting with the solution architect, tech lead, product owners and business analyst can get very heated discussions, as the solution is being challenged.<br><br>There is a tension in the scope of the solution. It used to be an MVP, however, this would be dirty work that was not scalable and needed to be thrown away after presentation. However, the organisation is moving away from this and developing MVP's that can be built upon.<br><br>Disadvantage is that the solution architects are a scarce resource and a bottleneck. There is more work than they can absord. This reduces the autonomy of the teams, as they run in impediments if the solution architect in unavailable.<br><br>Sometimes tech leads skip the conversation with the solution architect, for example by agreeing with another tech lead on a specific solution without consulting the PO or solution architect. Usually this is discovered only due to end-to-end testing. Getting this feedback faster into the bubble is a struggle. This is usually at the micro-level, but can have knock-on effect further down the line depending on the context due to dependencies.<br><br>It is difficult to work agile when the financial processes and targets are very waterfallish. | this on a structural basis, however these are edge cases. |
| Why are (dis)advantages related to the interaction and not architecture or agile alone? | Description | Not discussed. | The fact that you have an architect who doesn't talk to anybody or doesn't talk to PO's, what good does it do?<br>Right, so as an architect, you are there to in essence figuring out how to enable a scalable, maintainable and sustainable setup that will enable us to be successful in the future as well as today. Now, how do you determine where should that future go and where you had it and etcetera? By talking to many different stakeholders and product owners are being one of the stakeholders that you normally |

| | | | |
|---|---|---|---|
| | | | would interact with to figure out: all right was the direction that our business is taking, right? What is it that we need to be prepared for and want to, five years in advance because the strategy is going that way. But we as an IT team don't have a capability in place in order to accommodate for that, the same time the architects play a role to a bit guide that version as well, right. So, we often encourage what we call it driven innovation. We're saying, hey, do we have some fantastic solutions that we as a team can bring as their proposal to the business to actually drive innovation and improve strategic sort of competitive edge or maybe just optimize processes and automate things, which may be the product owners would have never thought of, because they are not aware that it exists or that sort of that's a possibility that's kind of where the architects would bring that innovative suggestion to the PO's. So, if you will say they are not talking to each other, then all of this disappears. Right. And you have two organizations that are sort of running side by side and not communicating with each other. So, it's again it's as simple as that sort of what is the power of communication. |
| Communication in combination | Irregularly / weekly / daily / Description | People do not always know when to raise their hand during a sprint, when to escalate or ask a question. | Both, formalised and ad-hoc meetings. So, we have structural meetings in place where we sort of we call them Scrum of Scrums but in this case on tech lead and solutions architect level. In our case we have a weekly call like this where all the architects and technical leads from my department look together into all the demand that is coming and they also do a bit of a, so they update each other on who's working on what gets changed in the landscape so that they can identify potential dependencies between each other on a larger scale. And they also present their solution design so they can do a bit of a peer review, and feedback has to write it doesn't sort of make sense or not or can we improve the process that happens that happens weekly again, as a structural setup but then you have many of the more ad hoc meetings, calls, etcetera that then not the whole group, but few individuals from the group would set up on a particular topic. |
| Impact of communication | Description | See interaction cell on bubbles. | Ability to identify dependencies on a larger scale, while also keeping the ability to discuss things that are related to a smaller set of stakeholders.<br><br>Ability to capture feedback early. |
| Knowledge of architect / team | Description | See interaction cell on technology driven teams. | Both of architect and developer teams, ranging from very senior to junior and in-between. However, |

| | | | |
|---|---|---|---|
| | | | architects are more high-level profiles than developers. |
| Uncertainty addressed (types of) | Costs / schedule / quality / description | New 'stuff' that teams do not have experience with. | Many |
| Impact of uncertainty on project. | | This inexperience leads to more new findings or surprises during the development process.<br><br>Epics and features cannot be pokered, but resources need to be planned in advance.<br><br>There is a big risk in implementing new features on the platform due to the current stage of technical debt on the platform. If you, do it right there are a lot of benefits, but if you do it wrong there is a big penalty. The commercial priorities of business do not allow for cleaning up in the platform and making way for big ticket items. | Not discussed. |
| Strategies to address uncertainty | Description | Daily stand-up to discuss the issue, follow up with relevant people outside of daily stand-up.<br><br>Retrospectives to do root-cause analysis.<br><br>Plan a Spike, to gather knowledge on the issue. A Spike does not deliver a shipable product, but provides learning as a team as added value.<br><br>See other measures in cell on flexibility.<br><br>Since features and epics cannot be pokered they are T-shirt sized: S, M, L etc., to do resource planning a quarter up ahead. | Two ways to address uncertainties: 1) up-front planning; dependency and risk management. Everything part of normal project management. 2) When the impact of uncertainties is lower, you simply proceed in a 100% agile approach, incrementally building and solving impediments.<br><br>Then there is a combination of both, as cases that use more up-front planning also use agile. However, in the fully agile cases the organisation takes more risk for low risk, low impact topics. |
| Governace structure/strategy | Description | Teams are very technology driven.<br><br>Strict project and architecture boards where impediments (concerning architecture) are discussed for which the 'stars and stripes' of those facing the impediments are not enough.<br><br>IT falls under the CFO.<br><br>Developers are all outsourced to another country.<br><br>Home brewn solutions are off the table, no longer developing solutions in-house. In the past solutions were developed in house. Since there was too much trouble in the past with upgrades and compliancy with home brewn software. | Agile transformation started more than 5 years earlier in IT than in business. The transformation to SAFe in business is still ongoing. Now if you look at how our both the governance of our organization was organized, but also how we have been working with a demand. A lot of it was what I would call vertical demand, meaning you can deliver it as a capability in one function or in one team or in one program. Therefore, you don't need too much of scalability across. What we see happening in the last several years already that most of our demand is shifting to become very horizontal demand, meaning you need multiple functions to really work together to deliver a solution end-to-end. And for that you need very effective planning, alignment sort of demand the portfolio management and so on for which SAFe seemed to be a smart solution.<br><br>Moving towards DevSecOps where engineers do all parts of the normal development life cycle. Meaning they develop, write test scripts, automate test scripts and deliver |

| | | | |
|---|---|---|---|
| | | | working software that is SAT, RT, Security and performance tested before releasing it into production.<br><br>Budgets are more rigid, planned for a year in advance. However, the demand shape is re-evaluated and adjusted on a continuous basis in an agile fashion. So, while the top-level budget is set for a year, there is still flexibility to prioritize and adjust scope within it.<br><br>For some projects there are fixed deadlines and is related to uncertainty, as it is the same with bigger risk, complex and high impact projects, that you would take more time to get deeper in the details because you want to be certain or have a strong chance of success to meet the deadline. For lower risk items you are less inclined to the deadline fix and more going with the flow of what the organisation needs at the time.<br><br>Compliance topics are present due to the industry. Compliance issues cannot be deprioritised and are simply done. Thus, a process is created where one is not surprised by compliance topics.<br><br>Purchases of tools or products go through an RFP, tender like, process.<br><br>In corner cases were architects and PO's do not come to an agreement on architecture or the feasibility of working under the architecture, there are several boards that can be escalated to, for example a consortium of architects, programme boards which include architects and DevSecOps managers. These issues could also be discussed in overall program performance status and programs.<br><br>If a project is not finished on time, then the costs can be absorbed, but in most cases there is agreement to limit the scope. So, time and budget are frozen, then the scope must remain flexible to accommodate that. Otherwise, you freeze all three, which is far from agile.<br><br>Normally there will not be compromised on the quality. |
| Impact of governance structure on project | Description | Platform has multiple teams assigned to it, to address the need for specific technologies. Project management is needed to coordinate the dependencies these different teams create.<br><br>The governance board helps to break through the impediment or escalates if necessary. | Mismatch in maturity of agile implementation within the organisation. IT is more agile than business. Many cases of a waterfallish approach with elements of agile as the organisation is in transition. The top-level stakeholders work more waterfall than agile. Gradually people move into a more agile rhythm. The mismatch in maturity creates |

| | | | |
|---|---|---|---|
| | | IT is seen as a cost centre and not as an added value. If they raise questions on the added value of the solutions they are asked to build, they are not heard and expected to build it anyway.<br><br>Developers have to pick-up calls in a different time zone. They have to fix issues, while already having worked a full day. There is a lot of attrition in teams due the long hours, this way knowledge leaves the company and technical debt.<br><br>Hard to cut out waste in development processes due to the distance and timezone differences.<br><br>Pull request system where code that has been written is going to be pulled on the branch, code and peer reviews to assure quality in outsourcing model.<br><br>More standard software solutions are used with customization, especially for standard processes. These should satisfy 70/80% of the requirements to cover the fundamentals. This allows to save the brains and capacity for non-standard cases that add value on the KPI's. | differences or complexities in communication and alignment.<br><br>In high risk, complex and high impact cases, you would also agree on the agile principle that you can fix your deadline, but your scope becomes variable. Not fixing the deadline means that certain requirements are being pushed, in theory indefinitely, because bigger or more important requirements are on the backlog instead.<br><br>Compliance does not just pop up, it is a normal part of the process, there are multiple steps and checks that are part of the normal development life cycle, including testing, quality and regulatory validation, legal checks etc. These processes span across several roles, for example the architects are aware of the compliance requirements and take them in as part of their solution, so do product owners. These are the non-functional requirements (NFR's). Since the industry is compliance intensive, there are specialised people who are part of a quality and regulatory group, who are validating every step of the way whether delivery is according to specifications, that the specifications are correct, and that testing is right etc. Protecting the organisation against compliance surprises.<br><br>For the RPF process, market research is done, where you go from a long to a short list of vendors, then a procurement process is started up. Vendors in the short list can result in a pilot or POC if the right legal and test agreements are in place. Risks with vendors are investigated and internal approval is needed, the level of approval that is needed is dependent on the tool level you buy. It is a lengthy process, you cannot just go and buy something. The length of the process depends on the complexity and size of the tool, but normally, starting from scratch it takes about three months to get a tool. If it concerns a large setup it can be much longer.<br><br>Limiting the scope means that there is agreement to create a lower MVP solution, that can be delivered within the available time and budget and still makes sense. The rest that does not get put up as an MVP will then become part of the future roadmap; phase two MVP or MVP plus. Whether that gets picked up is a future priority call based on budget availability and so on.<br><br>If the quality is not there, then in many cases the sessions will not be taken live into production until the |

| | | | |
|---|---|---|---|
| | | | quality is fixed. However, if it only considers a small edge case that is not working as expected but still functions, then a joint multi-disciplinary team decision may be taken to accept it. Depending on the frequency of occurrence. Still if it impacts the end consumer then in most cases there will be a no go for the release. If a small element is missing though, extra budget and time is usually not allocated to fix the issue. |
| Advise for those in a agile transition. | Description | Not discussed | Follow the process by the book if you are new to the process. Do exactly what the ceremonies prescribe even though sometimes you do not believe in it or that it is not adding value. Experience and learn how it should be and then modify to see what really works for you and what doesn't. But if you start the modifications immediately and have no clue yet how the real process should be working, you will end up thinking and working according to your old situation, since that is comfortable for you.<br><br>If your level of agile maturity is low in your organisation, employ an agile coach. Make sure that it does not become a side organisation, but in large organisations full time jobs for many people to really embed this agile mindset across the board.<br><br>Make sure there is top-down leadership support to actually work in an agile fashion and that your leaders get trained and onboarded into the agile way of working and know what to expect out of it, as one of the first steps. |
| Management | Description | Costs are saved on personnel.<br><br>Very senior level is very traditionally thinking. | See governance strategy. |
| Impact of management on project | Description | Co-located solution architects in the Netherlands are replaced by solution architects in the country where development is outsourced to.<br><br>Agile working is considered as working in sprints and that is good enough. This very senior level is hard to get behind the transformation, as they like to work waterfall with milestones. This is also present in other corporates: a hybrid situation between agile and waterfall, less so in start-ups. | See governance strategy |
| (Changes in) procedures | Description | We have a very toxic process also in our company where we handshake on enablers. So that the markets and the business can reach their financial targets. So, their sales targets are pegged to some enablers that we delivered as a digital team. This procedure is being changed, but currently it is still in use. | See interaction and governance strategy cells |

| | | | |
|---|---|---|---|
| | | To address the flexibility/fastness and stability tension, the outcomes of the session with the solution architect, product owners, tech lead and business analyst informs the involved stakeholders and later presents a set of options, including pros, cons and their preference, to the program board after having reached agreement.<br><br>The creation of the stakeholder bubble is dependent on the context. | |
| Impact of procedures on project | Description | Business does not care for stability, but IT is to blame when things break. There is no time to address technical debt as teams have to deliver new features. The values that are pegged to the enablers are very theoretical and sometimes inflated.<br><br>The program board will evaluate the set of options as well as their pros and cons and make a decision on which option to pursue. | See interaction and governance strategy cells |