Chapter 5

Exploring the Context of Architectural Decisions

“It’s tough to make predictions, especially about the future.”

- Yogi Berra


5.1 Introduction

In the past decade, the creation of software systems has changed rapidly. Traditionally, long-lasting waterfall projects (>2 years) were standard, whereas now rapid development (<3 months) with fast-changing requirements is becoming the norm for creating software products. In both cases, the architecture of the system has to be taken into account, although when, how, and who is responsible differs significantly. Formerly, experienced architects created models and documentation for the system beforehand, so the development team had a solid base of decisions on which to build. Nowadays, in more agile projects, the architectural decisions are made just in time by the development team itself, often assisted by a participating architect. Alternatives for heavy template-based documentation, like wikis or photos, are used to document the decisions.

This leads to a change in responsibilities, and in the role, of the architect. The responsibilities for the architectural decision process shift from the formal architect on one hand to the development team on the other hand. The architect takes on more of an advisory servant role within the project, often participating in the development team as a designer or developer. This difference, and thereby the newly needed alignment between agile and architecture, is the topic of this chapter.

In recent research, architecture in agile software development has been a topic of hot debate [2, 31]. While some authors emphasize the importance of architecture with Agile [45], others have described their experiences with Agile in product lines [5, 29]. This chapter contributes to the debate by presenting a framework that helps in identifying alignment problems in agile and architecture. Our framework is validated by case studies.
We have conducted a literature search on architectural decisions, the role of the architect, and how these decisions are documented. On the basis of this search, together with our own experiences, we constructed our Triple-A (Agile, Architecture, Axes) Framework. This framework identifies three different axes. The first axis describes the person making the decision (often called the architect). The second axis shows the way in which the architectural decisions are communicated (e.g., the artifacts used). The third axis describes the length of the decision feedback loop, the periodicity of a decision. We show that these axes can be used to profile a project or case study and that positions on these axes can be indicators for the success of a project.

The contribution of this chapter is twofold. First, we present a framework that helps project teams and software development organizations understand how they handle architecture. Second, we provide case study material that shows the effects when changing the architecture decision process. This helps organizations that are gaining agility identify what points of their architecture process need improvement.

The next section describes the research methodology used, followed by a description of our Triple-A Framework. Then our industrial cases are described. On the basis of our evaluation of the shifts in these cases and their resulting effects on business, the Tripe-A Framework is validated in Section 5.5. This chapter ends with related work, reflections on further research, and conclusions.

## 5.2 Research Methodology

We took the following steps to create the theory presented in this chapter. (1) During our participation in the industrial cases, we iteratively discovered changes in our case studies (for the good or the bad). (2) To emphasize the changes (described as shifts in Section 5.4), we have taken two points in time per case study and described the differences at each. We made the discovered changes explicit and categorized them. (3) Triggered by our experience while working on our cases, we conducted a thorough literature search of related projects and models. By generalizing our research and experiences, we created the axes that are the core of our Triple-A Framework. (4) We validated our framework with our case study material. (5) Finally, we identified several problems that occurred in our case studies and related them to changes in our model.

In our case studies, we used comparative multi-case analysis methodology [55]. Our initial theory building and measurement is done as an iterative process during the full period of all the case studies. We used longitudinal case studies for a period of time ranging from 9 to 48 months, where the qualitative data obtained by the participant observer complemented, in some cases, interviews with key participants in the project or product development team. In other cases the qualitative data was discussed with participants of the projects to validate the findings.

For each case, the research started with a discussion about what happened during the case, resulting in the descriptions in Section 5.4. Two phases were identified, resulting in shifts that were rated according to business impact. While some of the cases involved positive shift, other cases showed a negative shift between
5.3. The Agile Architecture Axis Framework

the phases. These shifts were used to validate our research-based Triple-A Framework. To determine the business impact of the shifts in the case studies, three success factors were used:

- The return on investment (profit minus investment cost)
- The speed of the project (whether the project progressed as scheduled and finished on time)
- The quality of the delivered project (whether the customer and the team were satisfied with the quality)

The return on investment is measured by the success of the projects: Did the project actually deliver? What were the investments and the cost? This evaluation was done after the fact and based on estimations of the researchers who participated in the project because first-hand financial data was not available to them. The speed of the project was measured by the speed of delivery of functionality (or changes in case of bugs) as perceived by the participant observer. The researchers based their quality assessment on discussions with customers and project team members.

The cases we use as a basis of our theory are not selected at random. From our experience, other cases could have been chosen. However, as stated by Eisenhardt [55], in case study research it is “neither necessary nor preferable to randomly select cases.” We have selected the cases that contained the clearest shifts, as described in Section 5.4. The cases are similar in that they all involved relatively small, collocated teams facing complex, real-life problems, but they involve a variety of situations — from a small product company (case Epsilon), to small projects at large companies (case Beta), to moderate-size projects at large customer sites (cases Alpha and Gamma), to a large company changing its way of working (case Delta). Three of the cases primarily involved the development of new products (Alpha, Beta, Gamma), whereas two involved evolution and maintenance of the running system (Delta, Epsilon) in addition to the creation of new functionality.

5.3 The Agile Architecture Axis Framework

In software architecture literature, many models are used to describe the software architectural decision process. For example, different models [23], templates [180], or ontologies [114] are used to describe architectural knowledge. Several authors have compared the ability of available models to process architectural design decisions. For example, De Boer et al. [23] describe a “core model” for architectural knowledge. They present a model and validate it through interviews and by testing it against existing models in the literature. Bu et al. [34] analyzed nine different approaches for describing design decisions. Such topics are beyond the scope of this chapter; however, we have seen that there are three essential aspects of the architecture creation process that are rarely thoroughly described

- The Architect. In architectural knowledge literature, the architect is referred to as the person responsible for the architecture, or the person who makes
the architectural decisions. However, who this person is and what his or her skills are, are rarely discussed. And the effects of these skills on the results of the project are also seldom written about.

- **Artifacts.** This term is often used as an abstraction of all things that are created during the architecture development process. Examples of artifacts that are mentioned are documents, models, or source code. However, the effects of the types of artifacts used on the outcome of a project are rarely researched.

- **Periodicity.** The “decision loop” [23] describes how decisions lead to new decisions based on the alternatives chosen. However, the periodicity of the decisions is rarely described. What is the length of time between the actual decision and the resulting validation of that decision in the quality attributes that the system needs to comply to?

The Triple-A Framework consists of three axes for describing where the decision process can differ in projects or companies. Each axis focuses on a different aspect of the architecture decision process as described above. In the following subsections, each axis is discussed by first describing relevant literature, followed by the points that we identified on the axis.

### 5.3.1 Who makes the Architectural Decisions?

Although there has been some debate about the role of the architect [118], in most literature concerning architectural design decisions or architectural knowledge the term architect is used but not well defined. For example, in the survey about architectural knowledge [23], the term architect(s) is used 11 times, without defining who this person is or mentioning this person in the described core model. In another survey [34], the term is only used three times, again without a definition or explanation. In [62], the work of architects is described, based on a survey conducted with a large group of architects. The authors mention that they included architects of different types in their research (“...including software, IT, solution, enterprise, and infrastructure architects”), but do not mention what the exact skills are or what effect a certain architect could have on the project. Kruchten [118] emphasizes that the architect has a broader role than just making the architectural decisions. In this chapter, we focus on the actual decision-making, not on the other things architects do.

Often, we have seen that people other than the formally assigned architects make architectural decisions. For example, the product owner (customer or domain decisions) and the development team (technical decisions) are heavily involved in, and sometimes responsible for, the architecture decision process. We have extended the scopes for architects described in [135] (Enterprise, Domain, Application) with two additional roles we encountered in our industrial cases (Management, Development Team). Note that the described roles are not one-to-one mappings to position names. They should be interpreted more as baskets for skills that a person with this role possesses.

- **Management:** Management can consist of company or project managers. Managers can have significant influence on the architecture decision process.
However, the main focus of management is on project properties (on-time, in-scope). Typically, management lacks knowledge of the actual technical background of the system or the customer’s specific functional demands. Because management often has a high position in the hierarchy of the organization, its decisions are hard to debate.

- **Enterprise Architect**: Enterprise or solution architects are typically responsible for the decisions at enterprise scope [135]. They often have a thorough background in theory and sometimes in practice.

- **Domain Architect**: Domain architects [135] are often customer-employed people who have thorough understanding of what the customer actually needs. The role of product owner is typically existent in Scrum [109] projects. This person functions like a domain expert, but is not necessarily part of the customer organization, and is often close to the team. Domain experts or product owners are often responsible for architectural decisions that have a high functional impact.

- **Application Architect**: The application architect [135] is typically an architect that also writes code as a team member. To make the right technical architectural decisions, he or she must have up-to-date knowledge. Typical role names that we encountered for these people are senior developer or technical architect.

- **Development Team**: The development team consists of the people involved in the actual designing, coding, testing, and deploying of the system. This includes people with architecting skills. The responsibility of the decisions lies with the whole team, in contrast to the previous roles.

Although often more than one role is involved in the decision process, in our experience there usually is one (sometimes assigned) role that has the formal or informal responsibility for making the decisions. The “who” axis describes who the main responsible person (or persons) is (are) for the architecture decision process.

### 5.3.2 What Artifacts are used to Document the Decision?

Architectural design decisions are often traceable to, or represented in, artifacts. Many authors use the term artifact [23, 114] to describe that the decisions have a representation somewhere. Some authors emphasize that the decisions themselves should be first-class entities or artifacts in the design [100], or provide templates to make the artifacts for design decisions [180]. As described in [79], knowledge can be shared personally (remembering, talking, etc.), or more formally (documenting, modeling). We have identified the following points, ordered from heavy to lightweight documentation approaches:

- **Mandatory Template-based Documentation**: These artifacts are architecture documents that have to be created because they are part of an offer, agreement, or project plan. Examples of these documents are the functional and technical design documents.
Chapter 5. Exploring the Context of Architectural Decisions

- **Facultative Template-based Documentation:** Rational unified Process (RuP) [117] has a very extended set of document templates that can be used in software projects. An example of an architectural deliverable is the software architecture document. Because they are not mandatory, it is easier to decide not to use one of the documents. The templates add to the unnecessary complexity of writing down architectural decisions. On the other hand, templates can help to make sure certain aspects of a design are not forgotten.

- **Ad hoc Documentation:** This documentation type consists of all the random documentation that is present in almost every software development project that does not comply with a template. It can be structured on a common share or (semantic) wiki [80], or unstructured by e-mail or ad hoc sharing. Because they don’t have to follow a template, they are typically quicker to write—but with the risk of forgetting important aspects.

- **Meeting Notes (sketches, photos, etc.):** Meeting notes, in written or visual form, can be used to create very lightweight documentation of design meetings or open spaces. This method has the advantages of being very quick and the fact that the details of a meeting are easier to remember when seeing the drawings from the meeting. However, it can be difficult for people who were not involved in the meeting to understand the decisions.

- **Direct Communication:** Direct communication is the tacit “documentation” that takes place every day. This can be in a chat, on the phone, or face-to-face. Direct communication is the richest form of communication because it is bidirectional (it is possible to ask for explanation) and multiple senses can be used.

Projects usually do not use just one of these communication approaches. However, there is usually one medium for reading and writing architectural decisions that tends to be preferred by the team. We use the items mentioned above as points on our “how” axis.

### 5.3.3 What is the Feedback Loop of an Architectural Decision?

During both the initial development and the evolution of the system, architectural decisions are made with a goal in mind. Often, these decisions are made to confirm nonfunctional requirements or the quality attributes to a certain level. However, sometimes the confirmation of the suitability of the decision takes a long time. Kruchten [118] describes antipatterns where architects are disconnected from the actual team. In our opinion, these patterns show what happens if the feedback loop from the decision to the actual validation of the decision becomes too long. Some authors have suggested the use of templates with “states” [114] or “status” information [180] to be able to document where the decision is in the feedback loop. However, neither of these representations of the state of the decision describes when a decision is actually implemented and validated.

To get a better idea of the validity of a decision, assessment methods like Architecture Tradeoff Analysis Method (ATAM) [107] are used to increase the confidence level for the decision. However, final validation occurs when the decision
is implemented in the system and is being measured by the usage. Typically three points in time are relevant for an architectural decision: the time the decision was made (“Decided” in [114] and [180]), the realization of the decision, and the validation that it was a correct decision (the latter two are nonexistent in the described literature). We take the time between the initial decision and the validation as elements for the periodicity of the decision.

- **Long (>6 months):** In long-running projects or offers, sometimes architectural decisions are made before the project starts. It can take several years before the decision is implemented and validated.

- **Medium (1-6 months):** Often, a proof of technology or Proof of Concept (PoC) is used to validate whether an architectural design decision has the desired result. These decisions are typically validated quicker than the long-running project decisions.

- **Short (<1 month):** In more agile settings, the validation of decisions can be much quicker. Especially when the technical infrastructure for continuous delivery is in place, the time between decision-making and validation can be decreased. The shortest cycle can be achieved during refactoring, where the decision is changed on the existing code base.

   Of course, some decisions cannot be validated in the short term. Also, a typical project has more than one decision point. However, projects tend to lean more towards one point on this axis, characterized by the organization style of the project. We use these three points in our framework as the “when” axis.

### 5.3.4 Summary of the Axes

To summarize, we have identified three axes that can be used to describe the architecture decision process in projects. In Figure 5.1, we visualize our Triple-A Framework in a radar plot.

In the following section, five industrial cases are introduced. All of the cases have a representation on the Triple-A Framework, which is evaluated in Section 5.5.

### 5.4 Industrial Cases

This section describes the industrial cases that are used in this chapter. In all of the cases, one of the researchers was involved as a team member within the studied company. As authors, at least one of us was involved in all the described industrial cases over an extended period of time. This allowed us to study the response of the case organizations to the changes that we saw. This chapter focuses on issues related to the work of software architects, although we studied the cases more broadly including other software development aspects. The cases are anonymized to protect the companies and customers involved. The roles that the researchers had in the cases varied from developer, architect, and team lead, to being part of the management of the organization.
For every case, the description starts with an introduction of the context, the customer, and the domain of the case. This is followed by a summary of project characteristics: the technology used, the type of architectural challenges that were tackled, the number of people involved, the duration of the project, and the process(es) used. Then a separate description of the two different phases is given. Every case description finishes by noting the shifts that occurred, and the business impact on these shifts. After the description of the shifts, the differences between the phases are summarized.

### 5.4.1 Case Alpha

Case Alpha involved the construction of a software system that had to replace a legacy geographic information system. The new system had to be coupled with several legacy back-office systems. The customer, a large harbor company in the Netherlands, initiated the project. The solution was service oriented, and consisted of several systems communicating with each other through an enterprise service bus (ESB). Most of the software was written in (Oracle) Java. This coupling was one of the most challenging issues in the project. This case consisted of a PoC and a realization phase, 3 and 6 months, respectively. Ten to twenty people were involved during the various phases of the project. In the PoC phase, a lightweight, iterative approach was used, whereas RuP [117] was adopted in the realization phase.
Phase One

During the first phase of case Alpha, an iterative approach was followed. The first deliverable of the project was a PoC that had to be ready on a predefined time-line. During this period, functionality was delivered every 2 weeks. In every iteration, new architectural challenges were tackled (e.g., how to process the data of the legacy system via the chosen ESB to the user interface, or how to correctly merge the real-time data from the ships’ locations to the static cargo data provided in one of the legacy systems). The customer was very happy with the result—a running technological PoC—and the supplier company was invited to participate in the execution of the next phase of the project.

Phase Two

In the second phase, the organization of the project radically changed. This was done because there was the potential that the project would need to be scaled up in team size. The result was that a group of eight architects from different companies was formed, assisted by a team of five project managers. They conducted thorough work in documenting all the possible situations, interfaces with other systems, etc. The development team that was involved in the initial PoC was rarely consulted, and was colocated on the other side of the Netherlands. The main results of this phase were the documents that were created: use case descriptions, architectural documentation, project plans, and more.

Shifts

The following were the most striking shifts that took place in case Alpha:

- Because of fear of making the wrong decisions, the focus was more on documentation and less on making working software.
- The iterations were longer (from 2-week iterations to half-year release).
- Because the system had to be connected to the deprecated systems that were running, there was a tendency to over-think the architecture to prevent making mistakes.
- There was no longer a structural feedback loop from the development team to the architects and customers and vice versa.

By making the decision process more heavyweight and focusing on documentation, the project was slowing down so much that it became paralyzed. After 6 months during the second phase, almost no working software was produced (one use case was realized). The project was discontinued because the customer no longer had confidence in it.
5.4.2 Case Beta

The product developed in case Beta was an administrative case management system for a department of the Dutch national government. This system had to handle (changing) regulations, and was to be used by various departments who had different demands for it. There was a multidisciplinary team of five to ten people involved for two periods of 6 months. The system was based on the Oracle Collaboration Suite, and the UI was developed with Java technology. The main architectural challenge in this project was the mapping of the desired functionality onto the technical infrastructure that was already in place at the customer site. The tradeoff between the specific solution and the generic components was also considered a major architectural challenge. The customer organization assisted the team with their operations and architecture teams. During the first phase of the project, an agile approach was used: high customer involvement, iterative delivery, and constant adaption of the product. In the second phase of the project, a more traditional approach was chosen, based on the Prince II [84] project management technique.

Phase One

Case Beta is split up in two phases that differed mainly in organization of the project. Phase one was facilitated by a very light RuP approach. The main goal of the project was to prove that case-based working, on an extendable technical solution, was applicable for the organization. There was a practical attitude towards documentation, and a focus on working software. Biweekly iterations were used to get the customer involved and to get feedback quickly. The results of this phase were a proof of technological validity of the solution and a first working version of the case management system. The customer was enthusiastic about the results and decided that a second phase should be initialized to complete the case management system for a specific department.

Phase Two

In the second phase, the team remained mostly the same, but the project management methodology changed because a new project manager was assigned. This phase was managed strictly in Prince II, by a project manager and a steering committee formed mostly of higher management from the customer and supplier organizations. The goal of this phase was to extend, customize, and implement the techniques from the first phase for a very small department in the organization. As prescribed by the development methodology used, the first aim of the project was to get the functionality and the architecture of the proposed system written down completely (functional design, technical design).

Shifts

The following shifts occurred in case Beta:

- In phase two, more emphasis was on (mandatory) documentation instead of working software.
5.4. Industrial Cases

- There was increased complexity of the organization around the development team.

- The architectural decisions needed to be made earlier in the design process, and the decisions were never validated.

Because the customer was afraid of missing something in the description and design, it took about half a year to complete the documentation. The results were so detailed and complex that the architect board that had to monitor the design was unable to determine if the resulting documentation guaranteed that the resulting software could be created. The project became paralyzed because of the amount of documentation generated. Therefore, no value was delivered to the customer in the second phase.

5.4.3 Case Gamma

Case Gamma was conducted at a medium-sized product company in the Netherlands. The project involved a new administrative software system for specific departments in Dutch hospitals. Changing regulations and different working environments needed to be taken into account from the beginning. The project was executed by a multidisciplinary team of seven people, assisted by the architect of the company. A Java stack (JSF, Spring, Eclipselink) was used to create this product from scratch, while a different team of approximately seven people developed a part of the back end separately. This separate development was one of the most challenging architectural parts of the project. Product development took place over 12 months. The team used Scrum [109] with biweekly iterations to show results while being able to adopt to changes in functionality.

Phase One

The company involved in case Gamma had the vision to create a reusable architecture for its products. This generic, reusable architecture was implemented in parallel to the development of the system. This generic software was also used by another product at the same time. Architects in cooperation with the company’s management made these architectural decisions (and the resulting interrelationship of the projects) before the project started. The development team was notified about the decisions, but the concerns they had were never seriously heard. During the first phase of the development of the system, the team felt that they had no influence on the decision process. Therefore, the atmosphere in the development team became less constructive. The reusable architecture was blamed for every defect in the system, as well as causing the system’s slow development.

Phase Two

The shift in this case came when the project team decided to take over responsibility and make the product independent of the decisions made before the project started. This included stubbing certain parts of the application, and sometimes even building functionality that was to be replaced by the generic software in the
(near) future. New decisions were made within the development team, and the company architect was kept informed but was not held responsible anymore. Because the connection with the other projects stayed, the validation of the decisions was still delayed, but the decision loop was shortened significantly.

**Shifts**

The following shifts were identified for case Gamma:

- The responsibility for the architectural decision-making shifted from the management and architect to the development team.
- There was a quicker architecture decision process because the responsible persons were always present.
- There was less dependency on other projects within the company.
- There was less architectural documentation, and the documentation used was more lightweight.

Although the change caused the project to take more time in creating functionality (some functionality had to be made twice—once by the project team and once by the platform team), the overall speed of the project enhanced (more functionality was produced), and the team was much more committed to the result and the product, which increased its quality.

### 5.4.4 Case Delta

Case Delta is a Fortune 500 company developing software products and services operating primarily on personal computers. The company’s products address both consumer and business markets and the company releases several products per year, including new releases of existing products and completely new products. The products developed by the company range in multi- to tens of millions of lines of code and tend to contain very complex components that implement national and international regulations. Although significant opportunities for sharing between different business units (BUs) exist, the company has organized its development based on a BU-centric approach. The products developed by each BU are typically based on a software product line. The company employs agile teams. It has new product development teams (who have no interdependency with other teams) and component teams for large established products in both North America and Asia.

The management and evolution of product architectures have been organized through architect teams that mentor and coach agile development teams. The organization arrived on this structure after falling into several traps around architecture and development efficiency. These traps included an overly complex software architecture for certain products, and software architects who had stopped coding and consequently had lost connection with the reality of software development that the team faced. The latter caused teams to be uninterested in the architectural
design decisions; the software architecture documentation and the system deviated rapidly. Finally, there were some signs of architecture work that was done for the architects, rather than for the benefit of the team or customer.

**Phase One**

In the situation of case Delta, the company used software architects as liaisons between general management, product management, and the development teams. The architects’ role was to translate business needs into top-level architecture decisions before development of new products or the next (yearly) release of existing products started. Because of their role, architects spent very little time actually building systems and would often make decisions based on outdated understanding of technology and the implemented product architecture. This resulted in architectures that were more complex than necessary, due to the need to bring the designed and real, implemented software architectures together in one system. A second consequence was that teams were not committed to the designed architecture, because it had limited bearing on reality and the architecture work was viewed as being done for the architects’ sake.

**Phase Two**

The organization realized the challenges and made four main changes. First, architects returned to coding and spent a significant part of their time developing software together with the teams. Second, the teams got more autonomy and interacted much more with customers, and system-level architects started to act more in a coaching and mentoring role. Third, the organization accepted that “life happens” and that it often is better to refactor the architecture of products when the need arrives than to try to predict everything beforehand. Finally, the latter also changed the perception around the importance of documentation and the organization focused much more on maintaining a stable product team that collectively holds the architectural knowledge in their heads.

**Shifts**

To summarize, the following shifts took place in case Delta:

- The architecture team got more connected to the development team.
- Customers were explicitly heard during the development.
- Architectural decisions were made more quickly, and were checked against the working software.
- Documentation was used less and more emphasis was put on the tacit knowledge of team members.

The shift in case Delta resulted in features that were better suited to the needs of customers, which was a significant benefit. In addition, because of agile development practices, customers could get access to features much earlier than in the
Chapter 5. Exploring the Context of Architectural Decisions

traditional development model. Finally, the significantly shortened feedback cycles resulted in higher quality of the overall system, because customers reported issues quickly and the team had the mechanisms to address their issues promptly.

5.4.5 Case Epsilon

A small startup company that creates a web-based product for the consumer market was the scene for case Epsilon. The project contained high-risk technological challenges; the architecture needed to be flexible in the beginning, to be able to handle the expected high number of users. The application was created in Ruby/Rails with a NoSQL back end based on MongoDB and Redis. The main architectural challenges were to be able to potentially scale up the application when lots of consumers are using the system, and being able to adapt the system to changing requirements from the customer. There was one development team of seven people that conducting (bi) weekly iterations (using Scrum) over a period of 12 months.

Phase One

In case Epsilon, the architecture was not fixed from the beginning, the main architect was an experienced developer, and the product owner had significant influence on the decision process. Architecture decisions were made before the implementation phase of any iterative development methods. The architect also helped the team with the coding of the software. After 3 months, the architect moved and was unable to participate in the project anymore, while the product owner got less involved in the actual development.

Phase Two

This started the second phase of the project, where no formal architect was with the team. All of the team members felt responsible for the architecture. No large architecture documentation was written — architectural decisions were made when needed and summarized in the wiki, or photos of meetings were shared in the internal chat. Important architectural issues were formulated as functionality and put on the backlog as user stories, often after debating them with everyone interested. Simplicity guided the architecture. The result of the project was a working closed Beta version of the product.

Shifts

The following shifts occurred in case Epsilon:

- Responsibility for the architectural decisions process changed from one architect to a team responsibility.
- There was a quicker decision process because the right people were always on-site.
5.5. Analysis

<table>
<thead>
<tr>
<th>Case</th>
<th>Domain</th>
<th>Period</th>
<th>Team size</th>
<th>Experience of team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>Harbor</td>
<td>9 months</td>
<td>10-20 FTE</td>
<td>Experienced team and architects</td>
</tr>
<tr>
<td>Beta</td>
<td>Government</td>
<td>12 months</td>
<td>5-10 FTE</td>
<td>Moderate team and architects</td>
</tr>
<tr>
<td>Gamma</td>
<td>Hospital</td>
<td>12 months</td>
<td>7-14 FTE</td>
<td>Moderate team, experienced architects</td>
</tr>
<tr>
<td>Delta</td>
<td>Product company</td>
<td>12 months</td>
<td>&gt;50 FTE</td>
<td>Experienced teams and architects</td>
</tr>
<tr>
<td>Epsilon</td>
<td>Consumer product</td>
<td>12 months</td>
<td>5-8 FTE</td>
<td>Moderate team, experienced architect</td>
</tr>
</tbody>
</table>

Table 5.1: Overview of Case Studies

- There was extremely lightweight documentation of architectural decisions and a focus on direct communication.

Although the change was not as severe as in the other cases, a definite change in the flexibility of the team was noticed. This resulted in quicker responses to bug or feedback reports and more predictable delivery of functionality (both quality aspects).

5.4.6 Overview

As an overview of the case studies used, Table 5.1 describes the characteristics of them all: the period, the team sizes, and the team experience.

In the following section, the case studies are mapped against the Triple-A Framework, followed by an analysis of the problems that can be generalized from the case studies.

5.5 Analysis

In this section, we will provide an analysis of the results by correlating them to problems that occur in software development. First, we will show how the cases can be mapped to the Triple-A Framework. From this, we will analyze what problems have been addressed in our work.

5.5.1 Mapping the Cases to the Triple-A Framework

To illustrate the value of the Triple-A Framework, we show that the changes in the cases from Section 5.4 can be seen as shifts among the axes of the Triple-A
Framework. In this section, we identify what effects shifting along the axes have on the efficiency of the whole case.

All five cases have been mapped on the Triple-A Framework. In the Appendix, a visual representation of the mapping is given. The results of the cases are summarized in Table 5.2. In this table, the cases are described and the shifts on the axes are visualized by: “-” shift toward the center, “+” shift away from the center, and “++” a radical shift away from the center. The results are split into the three parts that were discussed above: Return on Investment (RoI), Speed, and Quality. In the result columns, the “-” means that the result was worse in the second phase, and the “+” means the results were better in the second phase. Empty fields indicate no difference has been found.

Generally, we saw that the shifts towards the center of the Triple-A Framework (cases Alpha and Beta) increased complexity and tended to decrease productivity and speed. In the other cases (Gamma, Delta, and Epsilon), the shift to the edges of the axis resulted in increased quality and development speed.

### 5.5.2 Identified problems

As we have seen in our cases, organizations seek to improve agility because of real business benefits that can be achieved. Companies invest in software architecture and software architects for the same reasons. Unfortunately, as we discussed in the introduction, these areas are occasionally combined in ways that cause the organization to fail in its ambitions. From the experience of the described cases, we have identified six main problems. They are discussed below. With every problem, we indicate which cases involved the specific problem, what axes are affected and a short description of the identified problem.

**Architects as single point of failure**

Cases involved: Alpha, Gamma  
Axis affected: Who

The architect(s) often represent a single point of failure. The architect has to be the one that (a) talks to the customer to understand the vision and the most significant requirements, (b) creates the main structure of the system, (c) makes sure the solution is future-proof — especially concerning the nonfunctional requirements — and (d) makes sure the development team creates software conforming to the architectural decisions made. These tasks are hard for one person to do, especially
in larger projects. A project being dependent on one person presents a high risk.

**Complexity instead of simplicity**
Cases involved: Alpha, Beta, Gamma, Delta
Axes affected: Who and When

Architects are assumed to be the cleverest people in the team; therefore, they often create smart solutions that are more complex than they need to be to solve the problem at hand. The pressure from management and customers on the architect to create a “future-proof” architecture often enhances this effect. When groups of architects get too large compared to the rest of the team, the results look nice — but sometimes they are hard to implement. This is a typical example of the “create a perfect architecture, but too hard to implement” antipattern described by Kruchten [118].

**Outdated software architects**
Cases involved: Alpha, Gamma, Delta
Axis affected: Who

Often, the architectural decision-makers are not involved in developing the software anymore. This creates a lack of hands-on experience in the technology they are designing for. Because of this, their decisions are based on outdated assumptions and experiences, and because the decision-makers have no direct experience with any important design flaws, there is no incentive to change the design when necessary.

**Uncommitted teams**
Cases involved: Gamma, Delta, Epsilon
Axis affected: Who

If many important decisions are made solely by the architect(s), the development team does not feel primarily committed to the decisions made. If this happens, there is a lack of support for the decisions. In the worst case, the team opposes the decisions made and undermines the actual development of the system. This is a typical consequence of what Kruchten calls the “architects in their ivory tower” [118].

**Static architectural decisions**
Cases involved: Beta, Epsilon
Axis affected: When

As customer demands, technology, and organizations change, architectural decisions also need to change. Therefore, architectural decisions do not last forever. During the development and evolution of a system, architectural decisions are made and revised constantly. As architects in traditional settings are involved primarily in the earlier stages of development, the tendency is to make decisions earlier on and to keep to them for a long time. This makes it hard to adapt the
system to new challenges.

**Illusion of documentation**
Cases involved: Alpha, Beta, Delta
Axis affected: How

It is very tough to have good documentation when it is used as a communication medium. Often, documentation is out of date and is badly read. Since architecture documentation especially needs to be created manually, it is typically outdated within weeks—if not days or hours. In addition, very few people, even architects, actually read architecture documentation. This is both because of its obsoleteness and because it fails to help the reader build a relevant understanding of the system. However, these symptoms are rarely acknowledged—and when things don’t go as planned, often more documentation is mandated.

### 5.5.3 Summary

The description of the problems above can help teams identify problems and understand which change (shift on a axis) could help improve the project. Based on this set of problems, we can conclude that the who axis affected four problems, the how axis affected only one problem, and the when axis affected two problems. This is an indication that the who axis could be the most influential one.

### 5.6 Reflection

This section reflects on the findings from the previous section and discusses questions on the validity of this research.

#### 5.6.1 Findings

As with many solutions in software engineering research, the Triple-A Framework proposed in this chapter is no silver bullet. Often, the situation greatly influences the possibilities of companies, projects, teams, and individuals. However, the described axes can be used to see how responsibilities, timing, and communication methods can influence the results of a project. And, if a project or organization needs to change, the Triple-A Framework can help in identifying where the change can be made, by analyzing the current decision process and focusing on one of the axes where change can be achieved.

Some of the changes that derive from the Triple-A Framework have a big influence on how a project is run. When a project stops using certain templates, makes other people responsible for the decision process, or waits for design issues to occur, there is always the concern of trust involved. We have experienced that trusting people to do the right things is often very tough, especially within large organizations or large contract structures (as seen in cases Alpha, Beta, and Delta).

In traditional organizational setups, architects come in three archetypes that, although overlapping in some cases, have different responsibilities. The first type
5.7. Related and Future Work

of architect acts as the bridge between the business strategy and customers on one side and the software development team on the other side. Secondly, with highly complex systems, architects often have the responsibility for the end-to-end quality requirements of the system and coach individual engineers to make sure that new features and functionality do not violate system properties. Finally, some organizations share responsibility for the team between a project manager focusing on milestones and people management and an architect who acts as the technical lead for the team. In our experience, and this is the position that we take in this chapter, there is a fourth archetype. In this type, the architect becomes the coach for the development team responsible for facilitating architectural decision-making. In an age where teams become increasingly self-selected, -directed, and -managed [7], it is important that architects move away from traditional hierarchical, formal leadership roles and adopt meritocracy-based leadership styles. This can be done in an iterative way by accurately shifting along the axes of the Triple-A Framework.

5.6.2 Questions of Validity

Several matters raise questions on the validity of our research. First of all, the participant/observer method does imply some subjectivity. The results are qualitative (not quantitative), and based on the experience of the researchers participating in the project. However, this is one of the accepted ways to gather case study material in software engineering research, and all of our cases involved real-life industrial software projects that could not have been studied in any other way.

Although we have identified three axes that influence the results of the case study projects, it is possible that these are not the only parameters affecting the results of the cases studied. Due to the nature of our research, and the fact that it was conducted in real industrial settings, it is likely that there were other factors involved. However, we have seen that in all of the cases the shifts did occur in the same direction along the defined axis.

We have used only five cases, which involved mainly small project teams. As such, the degree to which this research can be generalized is restricted to projects of this type. However, as seen in the context of case Delta, the axes also show impact in larger organizations.

5.7 Related and Future Work

In the research community, there is currently a debate on the usefulness of agile software development. The trend is to say that there are enough stories of successful and failed projects done with various methodologies, but insufficient (empirical) evidence to found a conclusion [31, 150]. This chapter contributes in this debate by describing additional case study results.

There has been much attention given to documenting software architectures [42, 90], as well as documentation templates [117] and computational modeling for documenting relevant architectural knowledge. Recently, there has been

1https://www.omg.org/spec/UML/
a trend toward using semantic wikis [80], and some research experiments show promising results [73].

Another topic that is being discussed is the role of the architect [62]. Here, often the architect is responsible for creating and maintaining the architecture documentation. In this chapter, we have shown the importance to collaborative multidisciplinary decision-making of identifying who makes the decisions in projects.

In the architecture design decision research hierarchical structures are used to model architectural knowledge [45] or design decisions [100, 189]. This research often emphasizes the recording of decisions and the extraction of decisions later in the development process. This chapter focuses on the decision process itself.

The agile community often doesn’t explicitly describe the role of architects’ architectural documentation or architectural decisions when explaining what they do [109, 155]. Although there have been brave initiatives for merging the two [45], most of the agile community still has an aversion against architects and architecture documentation. Some authors emphasize the importance of architecture even in agile settings [115]. In this chapter, we have shown changes for software architecture in agile software development.

In future research, we would like to extend our validation of the Triple-A Framework to other industrial cases, specifically to larger-sized projects and distributed settings where direct communication is more complicated. Second, we will conduct more research on other possible axes that influence the architecture decision process. Third, we would like the Triple-A Framework to be based on a more discrete scale, to be able to score teams or companies.

5.8 Conclusions

From our industrial cases, we have seen a trend take shape. First, we have seen that leaning heavily on architects as the persons that should solve the architectural problems leads to unproductive teams, or even no working software at all. Second, we have experienced that when the focus is too much on large (architectural) documentation, the speed and quality of the project decreases. Third, we have seen that making important architectural decisions early on in the project leads to architectural problems during development. Although the agile community makes these claims, this is rarely backed up by case material. We contribute to this debate by presenting initial case study material.

In this chapter, we have followed two steps toward a better understanding of what happens around architectural decision-making. First, based on our experience and existing literature, we have generalized the Triple-A Framework for assessing how the architecture is handled in projects or organizations. Second, we have described five industrial cases, and we have identified shifts in these cases. We have shown that these changes can be mapped to the three axes that we created within the Triple-A Framework. We have seen that the successes and failures of the cases were influenced by the shifts that were made. These axes can be used to help teams that are becoming more agile to align their architecture practices.

From our research conducted at the five case studies presented in this chapter, we can conclude that moving on the axes of the Triple-A Framework influences the
success of a project. This means that by moving away from the center (development team, direct communication and short feedback loop), the projects became more successful (Gamma, Delta, Epsilon), while by moving towards the center (management, mandatory templates, long feedback loop), the cases became less successful (Alpha, Beta). We are planning to use our framework on additional cases to further validate our findings.

Appendix: Visual Representations of the Case Studies
Chapter 5. Exploring the Context of Architectural Decisions

Case Beta

Architect (Who)

Development Team
Application Architect
Domain Architect
Enterprise Architect
Management

Artifact (How)

Long (> 6 months)
Medium (1 - 6 months)
Short (< 1 month)

Feedback-loop (When)

Phase 1:
Phase 2:

Case Gamma

Architect (Who)

Development Team
Application Architect
Domain Architect
Enterprise Architect
Management

Artifact (How)

Long (> 6 months)
Medium (1 - 6 months)
Short (< 1 month)

Feedback-loop (When)

Phase 1:
Phase 2:
5.8. Conclusions

Case Delta

Case Epsilon