Universiteit Utrecht

Dynamics of Software Product Management & Software Architecture

Author:
Garm Lucassen
garm.lucassen@gmail.com
Student #: 3484572

First Supervisor:
Jan Martijn van der Werf
j.m.e.m.vanderwerf@uu.nl
Second Supervisor:
Sjaak Brinkkemper
s.brinkkemper@uu.nl

August 18, 2014

Abstract

Information exchange between software architects and product managers relies on simple communicative methods: long meetings supported by impromptu drawings. A counterintuitive premise when you realize software architecture has large, direct impact on product success factors: creating a winning product and delivering value to customers. This thesis analyzes the interaction between software architects and product managers to answer the main research question: *"How to support information exchange between Software Product Managers and Software Architects?"*. At the start of this thesis, we elaborately discuss literature on SPM, SA and their interdependence at software producing organizations. We argue that product managers and software architects contribute to each other's goals in four activities: (1) requirements gathering, (2) prioritization, (3) refinement and (4) transferring product context.

Subsequently, we conduct five case studies with to validate whether an interdependence between SPM and SA truly exists and to identify which activities SPM and SA collaborate on. The case study results confirm the existence of an interdependence between SPM and SA. Of the four activities identified in the literature study, requirements *gathering* and *refinement* are universal. Product managers require software architects' technical input to complement their primarily functional requirements list. While software architects require requirement details from product managers to be able to properly develop new features. Prioritization is common but the information exchange is trivial and one-sided. Surprisingly, none of the case companies collaborate in transferring product context. In subsequent research among 25 practitioners, we confirm that refinement and gathering are the primary collaborative activities for SPM and SA at SPOs.

The second phase of this thesis consists of design science research, itself comprising three stages. Based on the results of the case study, we present the Accurate Architectural Models Approach (AAMA) that engages both the product manager and software architect to ensure ongoing accuracy and timeliness. However, AAMA requires operationalization to achieve continuous model accuracy. With this goal in mind, we construct a simple wire-frame prototype for an automatic tool integrating AAMA.

Next, we evaluate the prototype by means of 9 case studies with a product manager and software architect from 5 SPOs. The most important feedback is that practitioners think product managers should not create architectural models, that the level of architectural detail is too high and that, despite these drawbacks, there is considerable value in being able to view connections between requirements and code components. Based on this feedback, we iterate and create a V2 of the prototype and evaluate it with an online survey. V2 gives users considerably more freedom to choose when and how to create architectural models and what detail level to apply per requirement. Unfortunately, although respondents think that individual features do offer value, the total benefits of AAMA do not outweigh the necessary extra overhead.

At least, that is what respondents believe based on a primitive prototype. Arguably, a real and working prototype will garner more positive results. On the other hand, we recognize that the presented solution incrementally improves the information exchange instead of solving concrete problems mentioned by practitioners. Consequently, respondents do not recognize or acknowledge that their processes need improvement, blinding them to the potential benefits. For AAMA to succeed, we first need to identify concrete problems that SPOs experience during requirements gathering and refinement and subsequently formulate specific solutions. Our next research efforts will focus on formally describing how requirements gathering and requirements refinement takes place in SPOs.

Contents

Chapter 1

Introduction

Numerous studies have shown that understanding stakeholders' requirements early in the software development process is imperative to project success (Nuseibeh & Easterbrook, 2000). Several authors posit that this notion applies to an early software architecture (SA) understanding as well (Nuseibeh, 2001; Rozanski & Woods, 2011). To support the evolution of SA and Requirements Engineering (RE), Nuseibeh (2001) adapted the *Twin Peaks Model* as introduced by Ward and Mellor (1986) which *"develops progressively more detailed requirements and architectural specification concurrently"* by developing *"requirements and architectural specifications concurrently"* while *"it continues to separate problem structure and specification from solution and specification"* (Nuseibeh, 2001).

The Twin Peaks model and the industry's perceived importance of requirements and architecture originate from research on software *projects*. However, the *product* software domain diverges significantly from projects. For one, as software product organizations (SPOs) grow, they encounter large numbers and varieties of stakeholders that generate a continuous input of thousands of requirements (Khurum & Gorschek, 2011). Consequently, decisions need to be made on which requirements to fulfill at what moment in time.

To manage these inputs and decisions, SPOs require special processes (Regnell, Berntsson Svensson, & Olsson, 2008; Wnuk, Regnell, & Schrewelius, 2009) that are typically the responsibility of *software product managers* [1]. According to Condon (2002), product managers are crucial to product and company success and Ebert (2007) even states that *"companies win or fail depending on their product managers"*. A product manager achieves product success by focusing on three goals: (1) creating a winning product and business case, (2) conquering markets and grow market share and (3) delivering value to customers (Ebert, 2007).

Because product managers are the primary interface and representative of stakeholders' requirements to the development team including the software architect, we postulate that successful collaboration between Software Product Management (SPM) and SA is a critical driver in achieving both goal 1 and 3 and therefore product success. Indeed, in (2006), Helferich et al. noted that SPM alignment with SA is critical to software product line success. However, few instrumentations support their collaboration and it is unclear *"how effective business-socio-technical congruence is achieved and what its empirically grounded business case is"* (Fricker, 2012).

[1] from here on: product managers

1.1 Problem Statement

This thesis aims to identify the relationship between Software Product Management (SPM) and Software Architecture (SA) by posing the following main research question: *"How to support information exchange between Software Product Managers and Software Architects?"*. The implied goal of our main research question is to develop an *effective* instrumentation that supports product managers and software architects in their collaboration. Before we can start this process, we first need to investigate and detail the complex SPM and SAM relationship. With this in mind, we subdivide this thesis in two distinct research phases which each encompass three research questions.

The first three research questions examine contemporary processes at Software Producing Organizations (SPOs), presenting a literature review and five case studies to identify SPM and SA's interdependence. In the second phase, we utilize input from the case studies to conduct design science research, itself comprising three stages. We create the Accurate Architectural Models Approach (AAMA) that engages both the product manager and software architect to ensure ongoing accuracy and timeliness. Envisioning AAMA as an automatic tool, we produce a simple wire-frame prototype and evaluate it in 9 cases with practitioners. In the final stage, we incorporate feedback from stage II into a V2 of the AAMA prototype and distribute a survey among practitioners to evaluate our work. The next subsections further detail the research questions.

1.1.1 Phase 1: Contemporary Processes

1. What literature is available on SPM, SA and their mutual responsibilities?

The literature review chapter starts by reviewing previous work on SPM and SA. Both sections discuss relevant definitions, constructs and theories. Next, we briefly present previous research on the overlap of SPM and SA.

2. Are SPM and SA at software producing organizations interdependent and if so, how?

By means of five case studies, we identify how product managers and software architects collaborate. We discuss what processes solve interdependencies and what information they exchange to do so. The research methods section details criteria to select the case companies.

3. What instrumentations do SPOs use to support information exchange between SPM and SA?

Aside from business processes and information exchange, we are interested in the instrumentations SA and SPM currently use in their daily activities. During the case studies, we ask them which processes and tools they use to facilitate their work. Furthermore, their evaluation of these instrumentations provide input to our requirements.

1.1.2 Phase II - Design Science Research

1. What are the functional requirements for an SPM & SA support tool?

We use the results from the first phase as our primary resource to formulate a new approach to ensuring architecture model accuracy. Based on this approach, we formulate feature requirements for a tool prototype.We accompany each with a short description of what it entails.

2. What is a viable functional design for an SPM & SA support tool?

We design wire-frames for each feature requirement based on their short descriptions. These wire-frames function as the tool prototype for the next research question.

3. Do product managers and software architects intend to use our tool prototype?

Next, we evaluate the tool prototype by leveraging constructs from UTAUT (Venkatesh, Morris, Davis, & Davis, 2003), a technology acceptance model to test users usage intention and behavior. In several case studies we evaluate the tool on three key constructs: performance expectancy, effort expectancy and facilitating conditions. Together, these questions will clarify whether product software professionals intend to use our tool prototype in their daily processes. Additionally, we will solicit improvement suggestions from respondents to gain further insight into their requirements.

Ultimately, the final research question provides the means to answer our main research question. Respondents usage intention degree will establish whether the tool prototype is a valuable addition to their daily processes, while improvement suggestions will further our understanding of what it takes to support SPM and SA information exchange.

1.2 Outline

The subsequent chapter details the research approach of this thesis. Next, we discuss related literature in the theoretical background in Chapter 3. Chapter 4 presents the results of our case studies. Chapter 5 discusses all the steps of our design science research. Finally, we summarize our primary results, discuss the implications of our work and present future research opportunities in the concluding Chapter 8. The analyses & tools presented in this thesis can inform product managers and software architects of methods to improve their information exchange.

Chapter 2

Research Approach

In this short chapter, we briefly detail the three research methods we utilize in this research: a literature study, multiple case studies and design science. Figure 2.2 presents a graphical representation of all the steps that comprise this thesis.

2.1 Literature Study

A solid understanding of contemporary literature on both SPM and SA is necessary to properly elaborate our research questions and formulate relevant case study protocols. Furthermore, although our instinct told us the relationship between SPM and SA is worth exploring, we could think of little literature that directly confirms this intuition. Therefore, our research starts with a study of literature on SPM, SA and their relationship in chapter 3.

2.2 Exploratory Case Studies

This research complies with Yin's (2009) three conditions to determine when a case study is appropriate for your research: (1) the research questions of our case study can be reformulated as *"How and why do product managers and software architects collaborate?"*, (2) we are unable to gain control of behavioral events and (3) we focus on contemporary events. To begin, we conduct five exploratory case studies at mature SPOs to build our initial theories relying on the case study approach by Yin (2009). In a later stage, we conduct Z shorter case studies at SPOs to evaluate our tool prototype, applying the case study method described in S. Jansen and Brinkkemper (2008).

Yin (2009) lists four validity tests that apply to case study research: construct validity, internal validity, external validity and reliability. Considering that S. Jansen and Brinkkemper (2008) adapt these tests in their approach, we discuss how we assessed these tests for this research. We use multiple sources of evidence to test construct validity, create a rich theoretical framework from previous literature and five exploratory, theory building case studies to ensure both internal and external validity and build case study protocols and databases (Appendix A) for reliability.

2.3 Design Science

Our thesis deliverable is a new artifact, which aligns with the goal of design science: *"Whereas natural science tries to understand reality, design science attempts to create things that serve human purposes"* (March & Smith, 1995). Design science consists of two basic activities, build and evaluate, which we approach according to the framework and its specifications as proposed by Hevner and Chatterjee (2010). This framework has been thoroughly tested as a vehicle for *"understanding, executing and evaluating IS research combining behavioral-science and design-science paradigms"* (Hevner & Chatterjee, 2010).

Figure 2.1 applies the framework to our research context. We conduct this research in an **environment** consisting of *people* that perform the product manager or software architect role in software producing *organizations*. Both conduct information exchange *processes* with one another, but would like to enhance their collaborative *capabilities* by improving *existing application technology*. Furthermore, the **knowledge base** *foundations* consist of existing *theories* on SPM and SA, contemporary, similar *instantiations* and *instruments* as well as *constructs* developed in an earlier case study. The environment and knowledge base generate business needs and applicable knowledge inputs for our design science **IS research**, which *develops* two artifacts: a tool design and a tool prototype. Upon delivery, experts *evaluate* the tool prototype in multiple case studies by leveraging UTAUT constructs as *methodology*. Based on the evaluation, we refine the design and prototype into a second, final version and re-evaluate it by means of a survey among SPM and SA practitioners, utilizing the same UTAUT constructs as before.

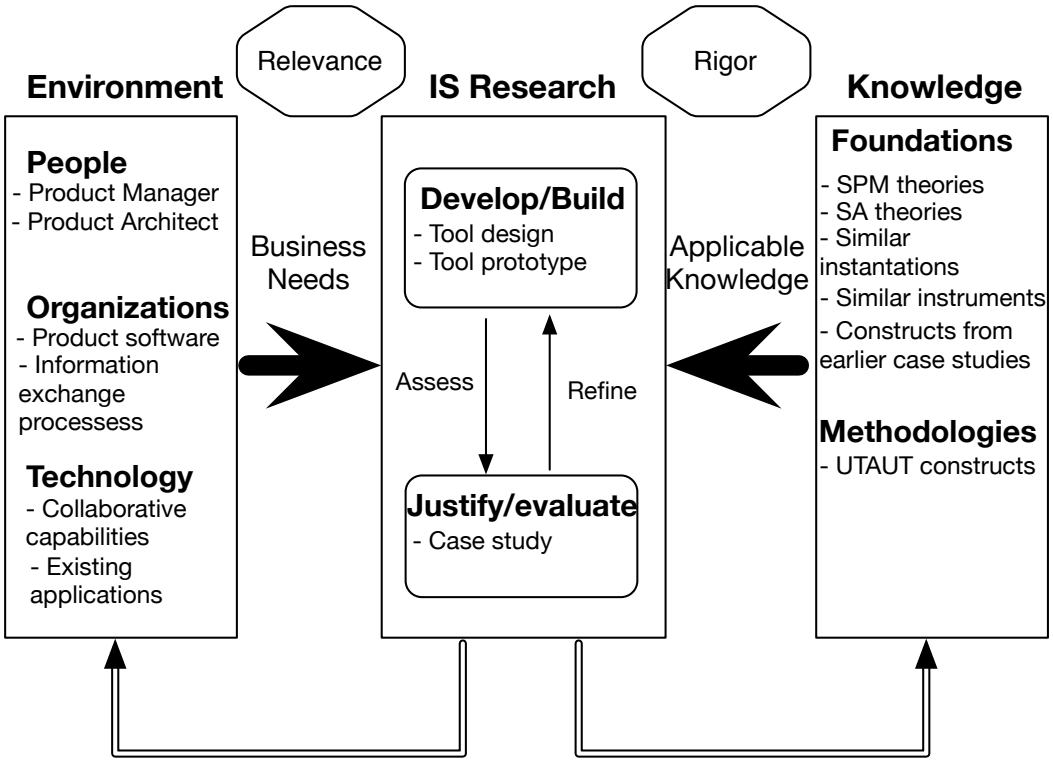


Figure 2.1: Information Systems Research Framework applied to this research

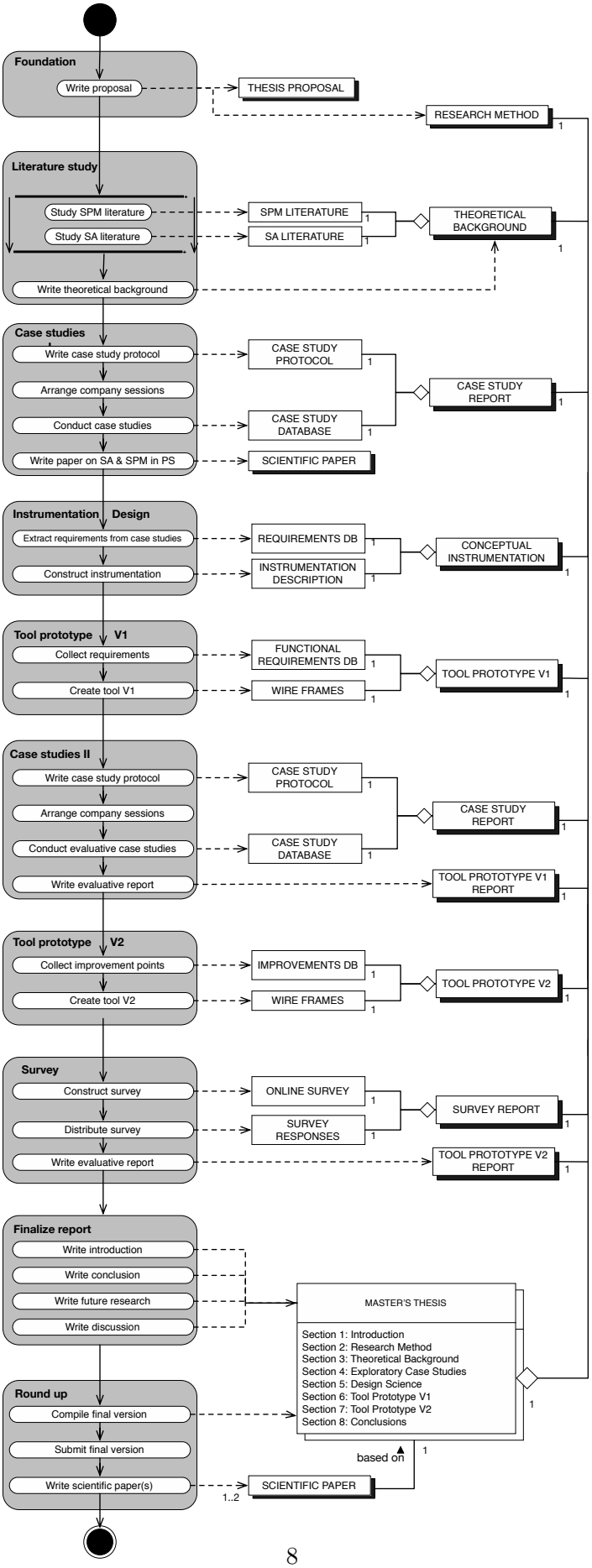


Figure 2.2: PDD depicting the steps taken during the creation of this thesis

Chapter 3

Theoretical Background

To support the evolution of software architecture and requirements engineering, Nuseibeh (2001) adapted the *Twin Peaks Model* as introduced by Ward and Mellor (1986). Shown in Figure 3.1, this approach *"develops progressively more detailed requirements and architectural specification concurrently"* by developing *"requirements and architectural specifications concurrently"* while *"it continues to separate problem structure and specification from solution and specification"* (Nuseibeh, 2001). Considering that SPM is closely linked to requirements engineering (Ebert, 2007), we believe that successful information exchange between product managers and software architects is critical to product success. In this chapter, we discuss existing literature on product manager activities, software architect activities and the relationship between these two roles in the following three sections. Based on our findings, we present four activities we believe product managers and software architects to collaborate in: requirements gathering, prioritization, refinement and transferring product context.

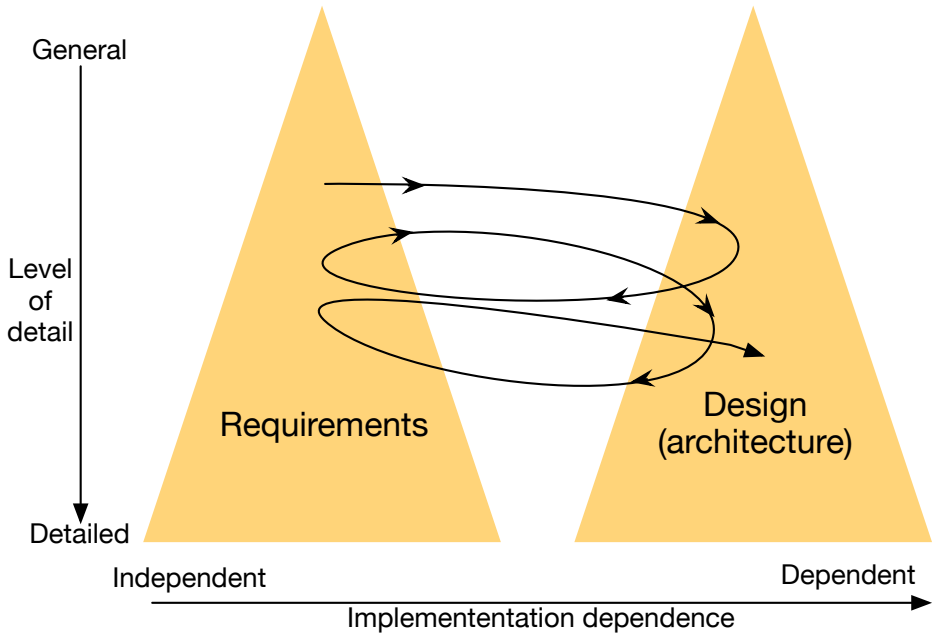


Figure 3.1: The Twin Peaks of Requirements and Architecture, recreated from Nuseibeh (2001)

3.1 Software Product Manager Activities

Software product management (SPM) is *"the discipline governing a software product over its whole life cycle, from its inception to customer delivery, in order to generate the biggest possible value to the business"* (Ebert, 2009). The domain defines a software product as *"a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market"* (Xu & Brinkkemper, 2007). As software producing organizations (SPOs) grow, they encounter large numbers and varieties of stakeholders that generate a continuous input of thousands of requirements (Khurum & Gorschek, 2011). Due to limited resources, decisions need to be made on which requirements to fulfill at what moment in time.

To manage these inputs and decisions, SPOs require special processes (Regnell et al., 2008; Wnuk et al., 2009) that are typically the responsibility of *product managers*. A product manager attempts to achieve product success by focusing on three goals: (1) creating a winning product and business case, (2) conquering markets and grow market share and (3) delivering value to customers (Ebert, 2007). He manages requirements, produces release definitions and defines products in an environment with many internal and external stakeholders (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal, & Bijlsma, 2006; Gorchels, 2000). This wide range of responsibilities makes product managers crucial to product and company success (Condon, 2002). Ebert (2007) even states that *"companies win or fail depending on their product managers"*.

Although the number of publications on software product management each year is growing, the field is still immature (Fricker, 2012). Consequently, SPM scholars are unable to provide scientifically validated best practices to industry. Nevertheless, several authors have developed SPM reference frameworks to exactly this end (Ebert, 2009; Kittlaus & Clough, 2009; Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010a). Of these, the Software Product Management Competence Model by Bekkers et al. (2010a) is based on extensive empirical research. Shown in Figure 3.2, it is a "comprehensive overview of all the important areas within SPM", consisting of four main business functions: (1) requirements management, (2) release planning, (3) product planning and (4) portfolio management. They are sorted by activity abstraction level and execution frequency. For a typical product manager, requirements management is a daily activity, while he only concerns himself with portfolio management every three months. Each business function consists of a number of focus areas that represent a strongly coherent group of capabilities. In theory, SPOs that hold these capabilities are more effective in their product management but this remains unproven to date. Together with the Situational Assessment Method (Bekkers, Spruit, van de Weerd, van Vliet, & Mahieu, 2010b), product managers apply the framework for process improvement in their SPO.

Vlaanderen, Jansen, Brinkkemper, and Jaspers (2011) present an approach to *agile* SPM, introducing product management sprints alternating with development sprints to ensure that requirements are ready for use when the software development sprint starts. During a product management sprint, the SPM team conducts *requirements refinement*: the process of refining requirement definitions that appear on a product backlog from coarse-grained concepts to fine-grained instructions that software engineers use to develop a product. Concretely, the product manager refines vision into specified requirements through three, progressively more fine-grained stages: themes, concepts and requirement definitions.

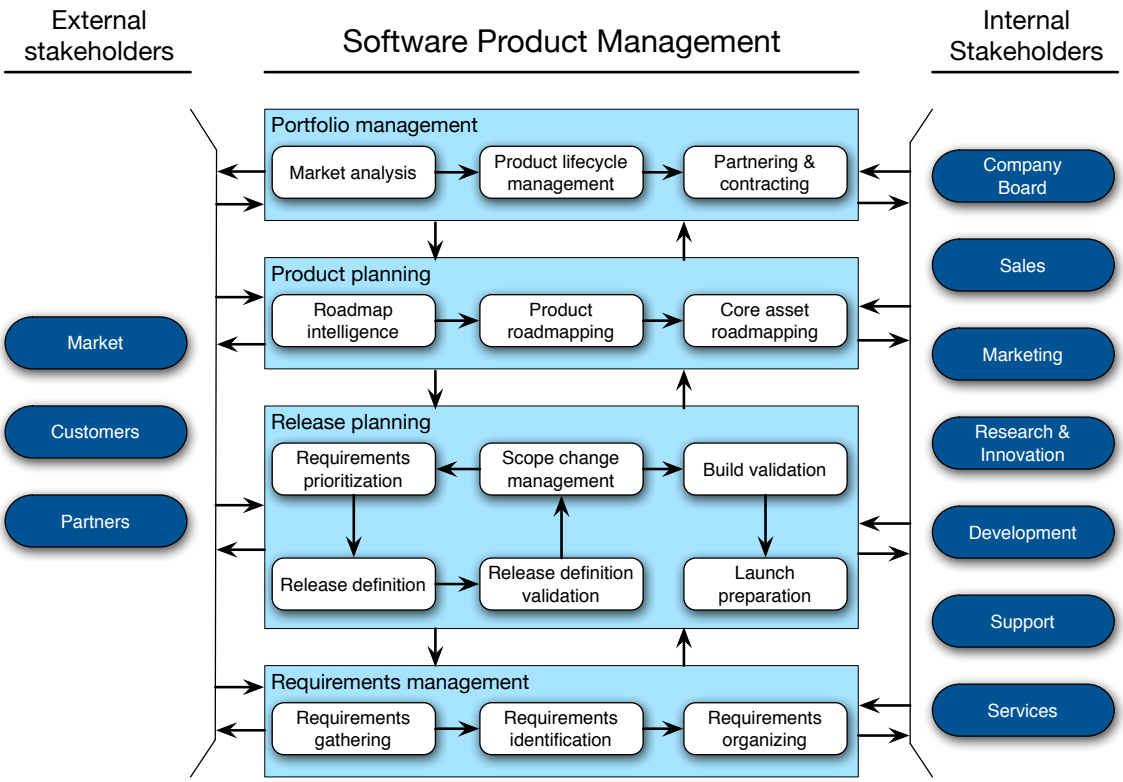


Figure 3.2: Software Product Management Competence Model, recreated from Bekkers et al. (2010a)

3.1.1 Requirements Management

Although *requirements management* (RM) is the most frequent, lowest level business function a product manager partakes in according to the SPM Competence Model, it is equally important for software *project* management. Because of its central role in software development in general, requirements management is the most extensively studied domain in traditional IS/IT research. Bekkers et al. (2010a) identifies three focus areas relevant in the product software context. They define these as follows: (1) requirements gathering: *"the acquisition of requirements from both internal and external stakeholders"*, (2) requirements identification: *"identifying the actual Product Requirements by rewriting the Market Requirements to understandable Product Requirements, and connecting requirements that describe similar functionality"* and (3) requirements organizing: *"structuring the requirements throughout their entire lifecycle based on shared aspects, and describing the dependencies between Product Requirements"*. These three activities combined with prioritizing requirements into roadmaps, releases and projects are traditionally called *requirements engineering* (Davis, 2005) and considered to be a core product manager responsibility (Gorchels, 2000; Ebert, 2007).

Requirements engineering typically occurs early in the lifetime of a project to prevent increased costs associated with requirements errors later on (Nuseibeh & Easterbrook, 2000). The product software context, however, presents specific challenges that necessitate a different approach to RE (Potts, 1995; Karlsson, sa G. Dahlstedt, Regnell, och Dag, & Persson, 2007; Sawyer, Sommerville, & Kotonya, 1999). The product manager encounters large numbers and varieties of stakeholders that continuously generate thousands of requirements (Khurum & Gorschek, 2011), making RM a daily necessity over the lifespan of a product. Furthermore, the large number of requirements frequently produces complications such as *information overload, combinatorial explosions* and *over-scoping* (Regnell et al., 2008).

3.1.2 Release Planning, Product Planning, Portfolio Management

The emphasis of this research lies on RM, because it is the most frequent activity a product manager conducts. Therefore, this subsection details the remaining product manager activities only briefly.

Release planning primarily concerns itself with software release management or *"the process through which software is made available to and obtained by its users"* (van der Hoek, Hall, Heimbigner, & Wolf, 1997). The broad business function comprises a variety of tasks ranging from prioritizing and selecting requirements to preparing the launch with internal and external stakeholders through negotiations and trainings.

A product manager conducts *product planning* to construct an action plan for a specific time period or a roadmap. The business function consists of three activities: roadmap intelligence, product roadmapping and core asset roadmapping. Information gathered in the first activity is input during actual roadmap construction.

Portfolio management focuses on *"strategic information gathering and decision making across the entire product portfolio"* (Bekkers et al., 2010a). Decision support information gathered during market analysis contributes to the business's decision making on product lifecycle management and partnering & contracting.

3.2 Software Architect Activities

A software architecture is "The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both" (Clements et al., 2002). Typical software architecture consists of a collection of views, which are "a representation of a set of system elements and the relationships associated with them" (Clements et al., 2002). An organization creates architecture documentation for multiple purposes. Organizations use it to design, analyze, communicate and/or educate a software system from multiple, varying stakeholder role perspectives. Considering that stakeholders can be as diverse as a customer and a developer, the seminal standard on Software Architecture (IEEE Standard 1471-2000), declares that architecture descriptions are inherently multi-viewed (Maier, Emery, & Hilliard, 2001). Thus an architecture is likely to comprise multiple consistent view collections.

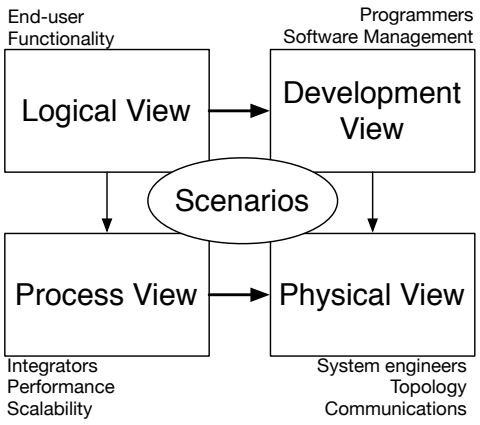


Figure 3.3: The 4+1 View Model of Software Architecture, recreated from Kruchten (1995)

Over the past decades, a staggering amount of architecture frameworks have been developed, most of them comprising multiple views. The ISO Survey of Architecture Frameworks currently lists 64 (ISO, 2011b). Of these, the 4+1 view model by Kruchten (1995) is the most popular multi-viewed approach. Shown in Figure 3.3, it breaks architecture down into specialized *logical, development, process* and *physical views* that inform specific stakeholders plus descriptive *scenarios* for discovery and validation purposes. The authors claim that the four specialized views contain all information that alternative views convey. A view typically contains multiple *architecture styles* which are a *"named collection of architectural design decisions that are applicable to a recurring design problem, parameterized to account for different software development contexts in which that problem appears"*(Taylor, Medvidovic, & Dashofy, 2010). A software architect has four primary tasks (Taylor et al., 2010):

1. Developing project strategy
2. Designing systems
3. Communicating with stakeholders
4. Being a leader

Over time, he develops a software architecture by making architectural design decisions (ADDs) based on the project context which consists of stakeholder requirements and previous ADDs (A. Jansen & Bosch, 2005). The ADDs in aggregate attempt to satisfy stakeholders' quality attribute requirements (Clements et al., 2002). Quality attribute requirements specify system operation, contrary to functional requirements which specify system behavior. Their importance for product success is equal. After all, a system's ability to produce correct information is useless when the system is unable to deliver that information in time, securely or at all (Clements et al., 2002). Achieving quality attribute criteria is central to the software architecture discipline. The ISO/IEC 25010 standard on system and software quality models posits that software product quality attributes comprise eight characteristics (ISO, 2011a) (shown in Figure 3.4):

1. Functional suitability
2. Performance efficiency
3. Compatibility
4. Usability
5. Reliability
6. Security
7. Maintainability
8. Portability

Each characteristic contains a number of related sub-characteristics, which "are useful for specifying requirements, establishing measures, and performing quality evaluations" (ISO, 2011a).

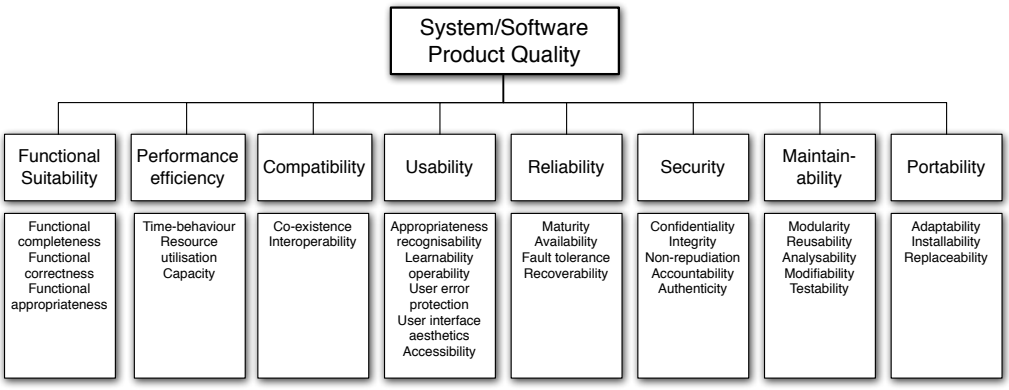


Figure 3.4: The ISO/IEC 25010 Standard Product Quality Model, recreated from ISO (2011a)

In the paragraph above, we saw that a software architect makes ADDs based on stakeholder requirements. Therefore, quality attributes are an indirect consequence of the project's set of requirements, creating a strong relationship between RE and SA. Drawing inspiration from Nuseibeh's Twin Peaks model for software development (Nuseibeh, 2001), several authors have formulated approaches to further relate software requirements and architecture (Hall, Jackson, Laney, Nuseibeh, & Rapanotti, 2002; Grünbacher, Egyed, & Medvidovic, 2004; Chitchyan, Pinto, Rashid, & Fuentes, 2007; Avgeriou, Grundy, Hall, Lago, & Mistrk, 2011). However, none of these are widely adopted in business or academia. For example, the CBSP (Component-Bus-System-Property) approach by Grunbacher (Grünbacher et al., 2004), which delivers a "proto-architecture" based on functional requirements to prescribe further architectural development, has been applied a limited number of times since its introduction (Vogl, Lehner, Grunbacher, & Egyed, 2011; Chen, Shao, & Perry, 2007).

3.3 The Software Product Manager and Software Architect

In the previous sections we introduced the software architect, product manager and their responsibilities. For both roles, requirements are crucial in similar ways. Yet, a limited number of previous literature discusses their relationship. Helferich et al.'s position that SA is critical to the success of a software product (line) Helferich et al. appears obvious, but no research to substantiate this claim is available. This section connects literature from both disciplines to formulate hypotheses on their relationship.

Of architects' four primary tasks, product managers contribute to two: *developing project (product) strategy* and *communicating with stakeholders*. While SPM has a strong functional focus, SA is primarily responsible for balancing the appropriate quality attributes (QAs). Although architects are able to formulate adequate technical solutions for any functional requirement, their proposed solution might not be optimal considering product context (Taylor et al., 2010). For example, the QA *performance efficiency* might be imperative for generating customer reports, while *reliability* has priority for the support interface. It is the product manager's responsibility to accurately convey context to enable the architect to make the right architectural decisions. Depending on the product manager's background (30-50% marketing or engineering (Ebert, 2007)) and position (R&D, marketing or P&L (Ebert & Brinkkemper, 2014)), the architect can discuss QA specifics with the product manager. Besides technical stakeholders, Taylor et al. (2010) notes that architects are in frequent contact with customers and users. For SPOs, however, the large quantity of diverse customers and users poses a communication challenge. Instead, the product manager acts as a representative on both sides. He communicates customer requirements to the architect, receives feedback and reports results to the customer.

Of the three product success goals introduced in section 3.1, the architect has direct impact on (1) creating a winning product (and business case) and (3) delivering value to customers. When the software architect is successful at his own primary tasks (1) developing a project strategy and (2) designing systems, the quality of the software product increases, consequently delivering more value to customers and improving the product's winning chances. However, what the impact is of a successful software architect in comparison to a less successful one is unknown.

Initially, we speculated that the product manager assumes the left peak of the Twin Peaks model. He exchanges functional details to refine requirements concurrently with the architect's specifications. In a previous case study however, the product manager's requirement refinement scope is limited: "The detailed definition of requirements is performed in three steps, of which only the first one is performed by the SPM team(s) [...] The software development teams then elaborate these [high-level requirement definitions] into requirements containing a detailed

description of some desired functionality, described in sufficient detail to work with" (Vlaanderen et al., 2011). The SPMCM by Bekkers et al. (2010a) does not recognize requirement refinement as an SPM activity to begin with. Instead, we believe there is a conceptual relationship between three activities: SPM, RE and SA. SPM is primarily responsible for high-level strategic issues such as market analysis, product roadmapping and fostering high-level requirements. RE refines the high-level requirements into sufficient detail to enable SA to realize the actual end-product. The information exchange between SA and SPM and their respective impact on one another's activities, however, is unknown. Figure 3.5 displays the conceptual relationship between SPM, RE and SA and illustrates this paper's central question: Are SPM and SA at software producing organizations interdependent and if so, *how*?

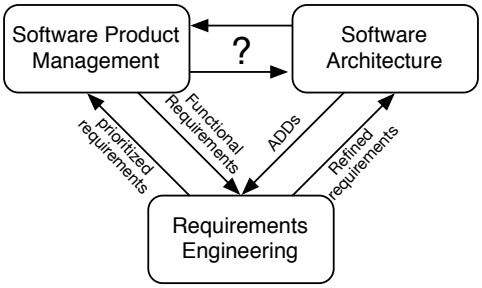


Figure 3.5: SPM, RE and SA relationships

Requirements' central role for both stakeholders is a recurring theme in this paper. Within the SPMCM, product managers involve SA during requirements gathering, identification, organizing, and prioritization. Architects require SPM input to refine requirements, formulate correct architectural approaches and communicate with external stakeholders. Both the product manager and architect have a common goal: satisfy customer requirements to deliver value and create product success. Reaching this goal demands selecting the right requirements for a release and enabling developers to implement requirements in the right manner. While organizing, identifying requirements and formulating architectural approaches are essential, we believe these activities are out of the stakeholder's respective scope. Based on these literature findings, we posit that product managers' and architects' contribute to each other's goals in four activities: (1) requirements gathering, (2) prioritization, (3) refinement and (4) transferring product context. In terms of the conceptual relationship in Figure 3.5, the product manager provides product context and requirement details to the architect in exchange for architectural expertise in order to enhance one another's decision making.

We elaborate on the conceptual relationship by taking the perspective of SPM and SA their primary goals as introduced in sections 3.2 and 3.1 (Figure 3.6). The goals are connected with artifact flows we believe product managers and architects to exchange. Note that you should not read these flows as a linear route, but as an indefinite, iterative process. The result displays the mutual contributions to their respective goals, which we detail in this paragraph. As the primary interface for internal and external stakeholders towards the architect, the product manager receives a continuous stream of market requirements and customer requests. He organizes and selects specific *product requirements* to present to the architect. The architect responds with *requirement feedback*, which the product manager communicates to the appropriate stakeholders with the intention to conquer markets and grow market share. On the second level, the architect transfers *architectural product knowledge* to the product manager to enable collaborative requirements gathering, identification, organization, prioritization and refinement. The result is a mature *release definition*, which the architect combines with *product context* to *develop a project strategy*. Next, the architect leads the development team to design the system,

making *architectural design decisions* based on the project strategy containing all provided inputs. These architectural design decisions lead to a finished product release and in turn deliver value to the customer.

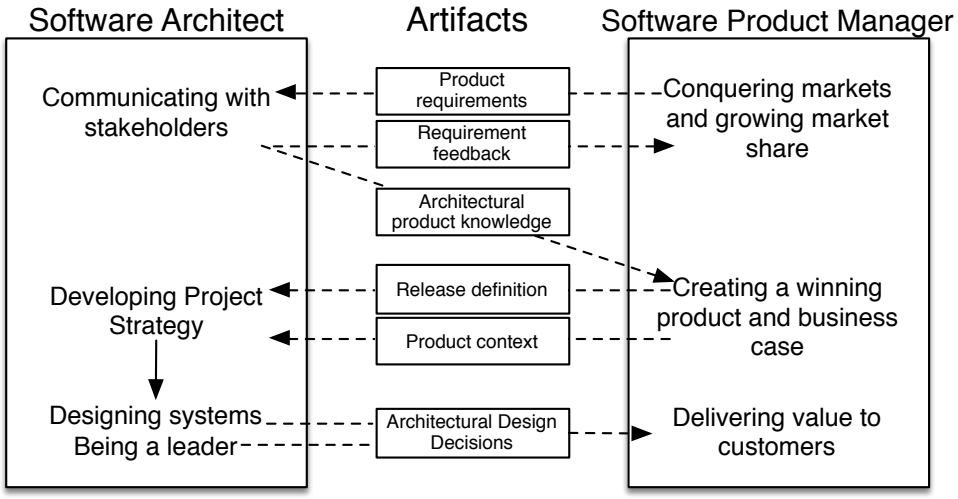


Figure 3.6: Reciprocal Contributions to SPM and SA goals

To support reciprocal information exchange, product managers and architects need architecture views that relate requirements and architecture by displaying information and data on the intersection of their respective responsibilities. Unfortunately, the approaches relating requirements and architecture mentioned in Section 3.2 do not satisfy this need. Their goal is to facilitate software architecture design by developing functional requirements models. Identifying a similar disparity, Salfischberger, van de Weerd, and Brinkkemper (2011) introduced the Functional Architecture Framework (FAF): an architectural view instrumentation that aims to support requirements management for SPOs with high requirement quantities. By linking requirements to specific architectural application modules, requirement responsibility can immediately be assigned to the correct individual or team. To use the FAF, the SPM must first describe the product's functional architecture with a graphical architectural description language called the *Functional Architecture Model* (Brinkkemper & Pachidi, 2010). It consists of a product context, a product scope that includes all high granularity application modules and typically two to three layers of sub-models. Next, the product manager assigns all existing requirements to the appropriate submodules and allocates newly incoming requirements in a similar manner. Finally, when a functional requirement enters development, the architect links his implementation solution enabling the product manager to analyze which functional requirements share a common technical component.

The subsequent chapter presents the results of five case studies at Dutch SPOs to investigate whether practitioners' activities adhere to these findings. For some activities, the information exchange will occur on an ad-hoc basis, while others will be formalized processes supported by creating models and specifications.

Chapter 4

Case Studies

This chapter presents results of our case studies into the collaboration between product managers and software architects at SPOs. The end goal of these case studies is two-fold: to identify (1) all collaborative activities among practitioners and (2) any instrumentations they utilize to support these activities. Although we had theories on the nature of their collaboration prior to the case studies, our intent was to explore as many options as possible. Due to this exploratory nature, a formally structured interview or standardized questionnaire was unsuitable (Corbetta, 2003; Kajornboon, 2005). Instead, we held semi-structured interviews to allow respondents to speak freely while maintaining the ability to compare responses. The interview contains questions such as: *"What are the essential aspects of SPM and SA collaboration in your organization?"* and *"Which instrumentations do you use to facilitate your information exchange?"*. After each interview, the interviewer puts together a case study report, describing the product manager and software architect's role in the organization, how both roles collaborate, whether either experience any problems in this collaboration and what kind of instrumentations they utilize. Based on the case study report, the interviewer records whether the respondents collaborate in the primary activities we identified in the literature study:

1. Requirements gathering
2. Requirements prioritization
3. Requirements refinement
4. Transferring product context

The five case study companies all originate from and are currently based in the Netherlands, but their size, age and target markets differentiate to ensure some degree of heterogeneity in the results. For each case study one interview of approximately 1,5 hours with both a (senior) product manager and (senior) software architect present took place. The following section discusses common, formalized processes in which product managers and software architects exchange information. The final section details the types of tools used to facilitate this information exchange.

4.1 Generic Findings

All case companies have similar approaches to SPM and SA. To begin, all claim to practice agile development, which prescribes a distinct (management) approach. The software architects in particular have loosely defined roles, which they grew into as the company and their seniority grew. They are the technical lead of a development team with the additional responsibility to ensure sound architectural decisions. As Taylor et al. (2010) puts it, the architect is considered "first among equals", leading the development team and communicating with relevant stakeholders such as the product manager. Due to the organizations' agile approach, software architects

only make ADDs before the start of a project for architecturally relevant requirements. ADDs for regular requirements are made on an ad-hoc basis. Identifying which requirements are architecturally relevant is done during requirements refinement, which each case companies conducts to some extent. Section 4.2 further details this collaborative effort.

None of the respondents declared that transferring product context is an important part of SPM and SA collaboration. Nevertheless, during our case studies we found that software architects have deep contextual knowledge due to two circumstances. As the product grows, the software architect's contextual knowledge organically grows along with it. When a software architect works on the same product for a long time, he is bound to establish an accurate contextual model. Furthermore, the software architects at 3 companies were part of the product's initial development team and thus had intimate knowledge of the domain, while the product manager role was introduced only after the product became successful. Respondents argue that these two factors substitute the need for contextual clarification. Software architects do rely on SPM to communicate with customers and users. Product managers function as intermediaries for external stakeholder requests that might concern the software architect. Support and sales first report to the product manager, who decides whether to escalate the issue to the software architect. Any resulting information or actions traverse the same paths. One product manager highlights the relevance:

> "If there's one thing that makes development including the software architect unhappy, it is when a customer calls eight times a day with new requirements or whether development is finished yet"

Contrary to what previous literature prescribes (Ebert, 2007), product managers at our case study companies typically also assume project management responsibilities. Furthermore, while each SPO gathers, identifies, organizes, and prioritizes requirements to create new release definitions, none had identical approaches. Despite this difference, all product managers independently identify and organize requirements. The responsibility and stakeholders involved during requirements gathering and prioritization diverge. Three case companies have formalized meetings where the product manager gathers requirements from the software architect. The remaining software architects transfer requirements on an incidental basis. One product manager conducts prioritization independently, only consulting the software architect when necessary. At 3 other SPOs the product manager relies on the software architect's input to make informed requirement decisions. The extent of this reliance depends on the product manager's technical product knowledge and capabilities. The following paraphrase succinctly illustrates the typical difference between product managers with an engineering and marketing background:

> Alice and I are product managers at FinComp. Alice has an engineering background, I have a lot of experience in professional services. Before the start of a sprint, Alice works out a first level of architecture and validates it with the software architect. I told the software architect not to even come to me with technical questions because I cannot provide answers. Alice can challenge architectures he comes up with, I cannot and thus rely 100% on the software architect to advice me on technical requirements.

One SPO has a special committee to collaboratively make decisions on requirements. Although the resulting broad stakeholder support is beneficial, the organization loses decisiveness and agility due to extra process overhead. The product manager and software architect proposed a solution:

> "We'd like a portion of the release to fully be our responsibility so we can autonomously decide to implement some product requirements regardless of current customer requests"

4.2 Important Overlapping SPM & SA Processes

In this section, we discuss the two activities during which our case companies' product managers and software architects exchange information during: requirements *gathering* and *refinement*. Gathering requirements from a variety of different stakeholders is crucial as a product manager (Bekkers et al., 2010a). Of all the internal stakeholders that contribute to the product manager's requirements gathering process, the software architect is unique. He is the only actor with the capability to identify technical debt and formulate requirements to solve critical deficiencies. One respondent that transitioned from engineering into product management clarifies the importance of formally requesting the software architect for input:

> "Gathering requirements from the software architect is our most important collaborative process. Due to my functional focus I no longer notice where code quality is degrading and overdue maintenance is growing."

The quote above illustrates that even product managers with a (strong) engineering background are unable to maintain in-depth technical knowledge of the product(s) they manage. This characterizes the first interdependence of SPM and SA: the product manager is responsible for scheduling requirements including technical maintenance, while the software architect has the domain knowledge to identify which product aspects need maintenance. To this end, 3 of our 5 case companies have scheduled meetings to discuss technical maintenance. The software architect presents which product sections require attention and why, the product manager shares his position from a business perspective and together they decide what issues to invest in. During these meetings information exchange is verbal, at times supported by simple informal models that happen to resemble the Functional Architecture Model by Brinkkemper and Pachidi (2010). Depending on the technical competence of the product manager, the two expend more or less energy on this communicative effort. At another case company, technical maintenance is a standard part of each sprint and the software architect's responsibility. This approach provides the software architect with more autonomous decisiveness and thus the ability to do his work more effectively. However, there is the added risk that the software architect focuses resources on the wrong issues from the product manager's perspective. For example, the unnecessary expenses to resolve an issue for a product feature that will be removed six months later can be prevented by coordinating technical maintenance.

Each of the case companies conducts requirements refinement, though none have adapted Vlaanderen et al.'s (2011) guidelines because they are unfamiliar with their work. Advanced concepts such as a product management backlog and alternating sprints with development have not been formally integrated into the organizations.Instead, the SPM team refines requirements on a case by case basis into high-level definitions. Later on at a formalized (bi-)weekly meeting, a product manager refines requirements with the development team and software architect in an agile, iterative manner. The product manager attends these meetings to coordinate and support the development team. He answers questions development poses, validates their interpretation of his work and might even lead the meeting. Different topics come up, at times including requirement validation and dependency linking; sub-activities of requirements identification and organizing.

Two product managers note that it is imperative to convey *why* they want to develop a feature. This enables developers to autonomously answer questions, preventing additional information requests during development and/or incorrectly implemented features. Requirements go through as many refinement iterations as necessary until the requirement definition has sufficient detail for development to start developing in the next sprint. Effectively, SPM and SA are going through the twin peaks stages until they hash out the lowest level of detail from the requirements

perspective. We posit that requirements refinement is the most important collaborative process during which product managers and software architects exchange information. The activity's formalized nature, relative high frequency of (bi-)weekly meetings and importance of the end result have a high impact on the product's quality.

4.3 Instrumentations Facilitating SA & SPM Information Exchange

All respondents recognize that they use a variety of tools, but note that few are applicable to this research context. For example, each SPO uses a tool to track features such as Jira, OnTime or Pivotal Tracker. These tools are central to their development activities, primarily benefiting short term project management instead of the more conceptual information exchange between a software architect and product manager.

Just one of the case companies claims to use a tool specifically to facilitate information exchange between technical and less technical roles: Enterprise Architect (EA). Although EA has a variety of features, this company primarily used its modeling capabilities to create both technical and functional models of their product which are connected by a data model. The product manager discusses their EA experiences:

> "EA improves [people] alignment and [communication] clarity. Recording ideas in models well enables us to discourse efficiently; explaining how, why and what you want becomes comprehensible. In turn, translating functionality and business requirements into technical requirements is now significantly more logical. Furthermore, because all stakeholders operate around the different models, alignment occurs far earlier in the lifecycle."

Despite these advantages, respondents from the other 4 case companies explicitly stressed their aversion to static documentation tools. Although none object to modeling itself, they warn against meticulously creating models for company-wide usage. Because, despite their meticulous effort, modeling is a human activity that *will* produce errors. While not a problem in itself, errors occur in every aspect of software production, colleagues perceive models as flawless representations of the truth which the typical model does not live up to. One respondent stated: "No documentation is better than outdated and thus wrong documentation", to which his co-interviewee, a product manager with limited technical background, added the following nuance: "If it's not a living document, it does not serve a purpose. If it's got a lifespan of X amount of time, don't invest in it. If you know it's a living document that you're going to refer to for the rest of your life, then nobody has an issue updating it.". Yet, neither could think of a document within their SPO with this kind of urgency.

Instead, these respondents create rough models, (functional, data, flow, etc) on a whiteboard when necessary. Aside from always being up to date, this approach permits hands-on collaboration on the whiteboard and communicative flexibility in what aspects of the model you draw, directing focus to only those parts that matter. Often, these models resemble the Functional Architecture Modeling technique (Brinkkemper & Pachidi, 2010), consisting of high-level elements that exchange high-level data objects. When participants deem a specific architectural approach is desirable, someone photographs the drawing and adds it to the requirement specification. The drawing then provides high-level guidance without becoming a fully thought-out implementation plan.

4.4 Preliminary Discussion

Case study analysis shows that all of the SPOs have some form of SPM and SA interdependence, typically addressed at a (bi-)weekly meeting. Although the formalization degree of these meetings varies, requirements are a common central topic. We found direct indications for three out of four collaborative activities presented in section 3.3: (1) requirements gathering, (2) prioritization and (3) refinement. Much to our surprise, respondents indicate that software architects do not need (4) product context clarification from product managers to identify the most important QA and formulate correct architectural approaches.

Negative indications	**Direct indications**
Product context clarification	Requirements gathering
Indirect indications	Requirements prioritization
Requirements identification	Requirements refinement
Requirements organizing	

Why do our respondents not recognize the need for clarifying product context to each other? The case study SPOs avoid having to communicate product context to new software architects by not hiring external recruits for architectural roles. Instead, they promote software engineers whose contextual knowledge already is close to on par with the product manager. Furthermore, the software architect receives sufficient contemporary contextual information in an implicit manner during his involvement with the product. However, we believe explicit context exchange is beneficial for the software architect's understanding in the long run. One software architect clarifies: "We used to be our own customers. Nowadays we are out of touch with what it's like to work in the domain. Because we haven't done the work in 15 years and the field has evolved drastically we need to test our assumptions with customers."

All other requirement related activities are part of the collaboration: (1) product managers request software architects' for input to his requirements prioritization processes, (2) the software architect has a unique contribution for requirements gathering and (3) they refine requirements into sufficient detail for development together. Additionally, during requirements refinement different topics come up including *requirement validation* and *dependency linking*. Through these two sub-activities, SPM and SA indirectly collaborate on *requirements identification* and *organizing* as well. Finally, the product manager acts as an intermediary between the software architect and customer requests and requirements.

Of all collaborative activities, requirements gathering and refinement are particularly interesting due to the complex, reciprocal information exchange that takes place. When a product manager gathers requirements from internal stakeholders, the goal is to include as many relevant perspectives as possible to gain insight into what issues are currently important within the organization. Unfortunately, there is no specific literature describing how product managers gather requirements, let alone how specific stakeholders take part in this activity. Without this information, formulating approaches or designing instrumentations to support SPM and SA alignment is challenging. Our case studies show that software architects contribute a unique class of requirements because the product manager is unable to reliably identify technical debt himself. Thus the software architect attempts to clarify the product's weak aspects in formalized meetings. However, he can only do so by conveying high-level technical details that are conceptually out of scope for the product manager.

The extent to which our product management respondents participate in requirements refinement is out of scope in a similar manner. In theory, development conducts requirements refinement to add sufficient detail to underspecified requirements, enabling each team member to fulfill any requirement in a later stage. However, 4 of the product managers participate

in requirements refinement sessions. In their experience, the independent, document-driven approach for refinement described in (Vlaanderen et al., 2011) generates too many requests for more details. Instead, software architects refine requirements collaboratively with product managers to resolve impediments as early and effectively as possible.

4.4.1 Implications

Looking back at the conceptual relationship between SPM, RE and SA in Figure 3.5, these findings enable us to formulate an answer to the second research question: "Are SPM and SA at software producing organizations interdependent and if so, how?" Product managers and software architects collaborate by exchanging information to gather QA requirements, collectively prioritize requirements and iteratively refine requirements selected for a release. Considering the large influence these activities have on delivering product value, we argue that effectively aligning them is a strong driver of business value. After all, a product is more successful and cost efficient when the organization gathers, selects and refines the right requirements in the right manner. To align SPM and SA in these activities, effective communication on a high technical level is a necessity.

One way to facilitate this information exchange is by creating models that both stakeholders are able to reason about and discuss. However, of our respondents, only one SPO utilizes formal technical and data models structured in a company-wide standardized manner. Although their experience with the tool is positive, all other respondents note one critical drawback: human-made models are outdated, wrong or both and using them will lead to corresponding decisions. These skeptical respondents rely solely on the most basic communicative tool, drawing on a whiteboard on an incidental basis. While they claim the benefits are great, they forgo SA's analytical and educational merits. We question the accuracy and timeliness of these high-level model sketches in practice. As product complexity and consequently the number of people involved in development grows, accurately depicting an SA from memory becomes exponentially more difficult. If neither formal nor incidentally drawn models are applicable, what approach should an SPO take to facilitate SPM and SA alignment?

Some respondents express the wish to automatically generate high level architecture views from data supplied by instrumentations they already use. A sentiment echoed in literature: "the most useful forms of documentation are views of the software that can be automatically generated" (Mirakhorli & Cleland-Huang, 2013). In the next chapter, we further explore the opportunity to automatically generate architecture from source code and connect requirements to all its elements.

Chapter 5

Design Science

As promised in the previous chapter, this chapter explores the concept of automatically generating architecture and connecting components to requirements. We elaborate on attainable opportunities in this space and conceptualize an instrumentation to support SPM and SA in their collaborative efforts.

SPOs face a dilemma in their efforts to align SPM and SA. Our case studies show that SPOs have two approaches to documenting a product: multiple, formalized models or no models at all. Over time, the former leads to outdated, inaccurate models that harm the organization, while the latter forgoes all benefits of models by relying on stakeholders communicative competence to exchanging the right information. We believe the industry requires an approach that utilizes models understandable to both stakeholders and ensures their timeliness and accuracy.

During our case studies, we noticed that none of the information exchange approaches by our respondents are satisfactory. Combining the literature review and results from our case studies, we identify four architectural model uses relevant for product software:

1. **Incidental models,** which stakeholders draw when the need presents itself during meetings
2. **Informal models,** that record incidental models for later reference if necessary
3. **Discussion models** depict the product on a high (functional) level to enable technical stakeholders to exchange information with non-technical stakeholders
4. **Structured models** take a formalized approach at depicting architecture such as data models for documentation purposes, possibly captured in tooling

Of these, discussion models (DMs) are particularly relevant for our purposes. They adhere to the requirements argued by Whalen et al. (2013) for aligning requirements with architecture: requirements are decomposed into fine grained specifications and organized in a similar manner to the system's architecture.

Unfortunately however, DMs have similar drawbacks to structured models. While Salfischberger et al. (2011) prescribe continuously monitoring the architecture and adjusting the FAF discussion model when it is no longer accurate, the lack of a structured approach to keeping DMs accurate reduces their utility. In practice, over time the DM starts to diverge from the implemented situation. The software architect, however, is confident the DM is accurate enough for the task at hand and chooses not to update it or forgets to do so after implementation. Meanwhile, the product manager re-uses the inaccurate model for new requirements. Three iterations later the DM diverges from reality to such an extent that it is useless and both stakeholders complain when they have to create a new one. Concisely, human error and/or negligence causes DMs to become outdated, wrong or both as well.

To prevent this scenario, case study respondents indicate they wish to automatically generate high level architecture views from data supplied by instrumentations they already use. Motivating that they require always up to date architectural views to be able to use them on a daily basis. A requirement that is difficult to satisfy when relying on humans. In an interview on his experience with architecture and requirements in the product software industry, Jan Bosch echo's this sentiment: *"the most useful forms of documentation are views of the software that can be automatically generated"* (Mirakhorli & Cleland-Huang, 2013). With this in mind, we can conceptualize a design for an automated FAF instrumentation.

Each of the SPOs we interviewed has three data sources that we can utilize: (1) a project management tool that registers requirements, (2) the product source code containing all architectural relationships and (3) code commit messages which provide context to new code contributions. By combining these, we imagine a system with three primary functionalities. First the system generates a FAM-like architectural view from source code. By applying Natural Language Processing to existing requirements' names and descriptions the system suggests links to specific modules to create an initial FAF. Next, stakeholders make adjustments where necessary to ensure the FAF corresponds with the project's current state. Now, whenever a new requirement arrives it is connected to the FAF in a similar manner. When a developer implements a requirement and tags the commit message with the requirement's internal code such as *AUTH-46*, the system validates whether the requirements was connected to the right module. The result is a dynamic, multi-tier view depicting the software architecture with different levels of detail for different situations, similar to the *accurate and interactive business process maps* for business process management as introduced by van der Aalst (2009). Furthermore, zooming in on particular architectural aspects enables architecture discussions to focus strictly on the relevant code elements. Note that this a conceptual design with a scope that is far too large for this thesis.

5.1 AAMA - Accurate Architectural Models Approach

Earlier in this chapter, we noted that the lack of a structured approach to assuring discussion models' timeliness and accuracy causes human errors similar to formal models. A problem that we believe we can solve by addressing a specific subset of the scope we defined above. In this section, we propose the Accurate Architectural Models Approach (AAMA) to prevent model divergence. The idea behind AAMA is to involve both product managers and software architects in the architecture modeling process. AAMA consists of three co-existing model instantiations that both stakeholders work on during four distinct process steps. However, before an SPO can utilize AAMA, the product manager and software architect first need to create an *origin model* based on the product's current situation. On this originating moment, the AS IS and SHOULD BE model instantiations are identical. Now, both stakeholders start altering the models by following the steps shown in Figure 5.1 and described in Table 5.2. After the first iteration, the cycle restarts when the product manager uses the newly updated and accurate SHOULD BE as input.

AS IS	The product's current situation as seen from the architect's technical perspective
SHOULD BE	The product's current design, consists of the current situation and all pending design changes based on new requirements
TO BE	Future design of the product based on new requirements to be included in a release

Table 5.1: AAMA's three model instantiations

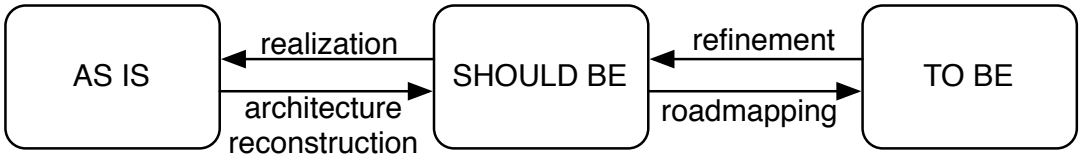


Figure 5.1: Accurate Discussion Model Approach

Roadmapping	When new requirements arrive, the product manager uses the origin model to create a TO BE
Refinement	In the second step, both stakeholders refine the TO BE to create a SHOULD BE
Realization	The software architect consults the SHOULD BE to realize the new requirements
Architecture Reconstruction	After realization, a new AS IS situation materializes which diverges from the SHOULD BE, prompting the software architect to conduct architecture reconstruction to warrant the SHOULD BE's accuracy

Table 5.2: AAMA's four steps

By applying AAMA, organizations enjoy three direct, positive consequences:

1. The frequent reviews and updates makes the DM a *living model* that multiple stakeholders verify. This prevents outdated models and over time corrects human made errors.
2. By ensuring that software architects as well as product managers examine and update the DM on a frequent basis, both gain a deep understanding of the recent changes to the product.
3. Having a high architectural view available on demand enables all stakeholders to create incidental models of strictly those product sections relevant to the discussion

We postulate that these consequence have meaningful, positive influence on the reciprocal contributions of SPM and SA. The mutual deep understanding and common view supports the frequent communication on product requirements and requirements feedback. Consequently, the software architect transfers the *right* architectural product knowledge more effectively and efficiently because he knows the extent of the product manager's architectural knowledge. Aside from time and cost savings when developing the product, this improvement increases the product's quality. Secondly, AAMA plugs into the Twin Peaks model by facilitating iterative TO BE DM refinement. The product manager expresses requirements in the TO BE DM, which the software architect examines for implementation and requests further refinement details from the product manager if necessary. The resulting release definition is more complete and unambiguous, contributing to the software architect's technical product strategy development.

Although AAMA prevents cumulative DM divergence from the current implementation, one major shortcoming remains in the product software context. During requirement implementation by the software architect, the product manager is already processing the new, continuously incoming requirements into a new TO BE DM. The moment that the SHOULD BE DM is updated, the product manager has already produced a new TO BE DM, possibly creating a divergence. This divergence will result in either the product manager having to alter his DM or the software architect to work with an inaccurate specification, reducing the DM's utility. How do we solve this issue? Some respondents suggest to automatically generate high-level architecture views from data supplied by instrumentations they already use. An idea supported

by literature: “the most useful forms of documentation are views of the software that can be automatically generated” (Mirakhorli & Cleland-Huang, 2013). Automatically generating a DM from source code and subsequently connecting all elements to requirements from the requirement database is an exciting concept. However, automatic architecture reconstruction to date has only a few incomplete successes for specific uses cases (Schmidt, MacDonell, & Connor, 2012; Fontana & Zanoni, 2011). The accuracy and relevance of an instrumentation that automatically generates architectural views is doubtful and the prospect of having the architect adjust the entire view manually is considerably less exciting. Instead, the following two chapters research the potential of automatically uniting the three different DM instantiations in AAMA to prevent divergence at any stage.

Chapter 6

Automating AAMA

In the previous chapter, we asserted that discussion models are appropriate for aligning requirements with architecture, but that the lack of a structured approach to maintaining their accuracy and timeliness reduces their utility. Our solution, Accurate Architecture Models Approach or *AAMA*, aims to ensure accuracy and timeliness. In this chapter, we present requirements for a tool that integrates this approach, construct a wireframe prototype and evaluate its merits with both product managers and software architects.

Because we consider an automatic AAMA tool as the next logical step for AAMA in itself, from here on we refer to the prototype tool simple as **AAMA**. The primary requirement for AAMA is to support each step: roadmapping, refinement, realization and architecture reconstruction. Additionally, the preceding chapters include many secondary requirements. With this in mind, we formulate the following requirements for AAMA:

Primary Requirements

Architecture modeler - before the product manager and software architect can start using AAMA, they need to create an AS IS representation of the product. AAMA includes an architecture modeler and lists to connect product requirements to its relevant architecture components. The produced model is central for the rest of the application.

Roadmapping - The product manager uses the modeler to add TO BE model fragments to the single model instantiation.

Requirements refinement - The software architect reviews the new model fragments to request more details from the product manager and modify the model if necessary.

Realization - The development team consults the product requirements list and associated model fragments to see how the new requirement implementation is expected to proceed.

Architecture reconstruction - After realization, the software architect verifies the implementation and, if necessary, adjusts the model to the new reality.

Secondary Requirements

A single model - Respondents remark that as their SPO grows, the number of models grows and their respective notation diverges. The resulting model fragmentation breeds confusion instead of the alignment stakeholders seek. To prevent fragmentation, we introduce a constraint: AAMA has just one architectural model that all stakeholders use in each step.

Zoom components - The single architectural model needs to be relevant for a broad range of stakeholders with both strong and weak technical knowledge. Enabling the application to zoom in on specific components creates different degrees of detail for each stakeholder's technical competence.

Integration Respondents emphasize the peril of static documentation tools which inevitably produce wrong and/or outdated architectural models. While we established that a fully automated tool is a Utopian dream, semi-automation can ensure satisfactory timeliness and accuracy for our purposes. This prompts us to discuss a crucial requirement: integrating AAMA with relevant other software an SPO already uses. AAMA will utilize the feature tracker, product code and its commit messages to recognize and respond to changes in requirements and the product code. For example, the application connects code changes to manual model adjustments and displays which code changes the software architect has verified.

In the next section, we guide you through the prototype tool that we created based on the requirements above. We present a wireframe for each of the steps in item 1-4 and elaborate on what a user is expected to do.

1. Create AS IS representation of the product
   (a) Import code
   (b) Model construction
   (c) Connect Requirements
2. Model details
   (a) inspecting a component
   (b) inspecting a requirement
3. TO BE representation
   (a) Import new requirements
   (b) Connect new requirements
4. Validate, adjust and review model
   (a) Consult the model
   (b) Code analysis
   (c) Model adjustment
   (d) Verification notification
   (e) Adjustment verification
   (f) Historical perspective
5. Feature roadmap
   (a) Requirements insight
   (b) Critical components
   (c) Filtering components
   (d) Clustering requirements
   (e) Quality requirements in models

To illustrate each of these functionalities, the wireframes are presented as running examples utilizing the sample application from the Ruby on Rails (RoR) tutorial by Hartl (2014). Ruby on Rails is a web framework that uses the model-view-controller (MVC) architectural pattern to structure programming. The clear separation of concerns permits easily creating clean architectural models and presenting varying levels of detail. Furthermore, the author has a year of professional experience with the framework, ensuring that the created models adhere to RoR's MVC style. The simple tutorial teaches RoR basics by guiding developers through the process of creating a Twitter clone. This clone allows visitors to create an account, share microposts and interact with other users. The running examples situate in chapter 11. In the preceding chapters, the developer has implemented functionality so users can login and share microposts with the world. The next step is to add the canonical feature of Twitter: following users. While the tutorial also includes code for unfollowing users and viewing a feed filled by users you follow, this example restricts itself for comprehensibility.

Finally, note that we developed the AAMA prototype for organizations that use an Agile development approach and apply a rigorous branching model similar to Driessen (2010). This model comprises a *development* branch which holds the most recent internal application version that passes all automated tests, multiple, concurrent *feature* branches that developers merge into the development branch when they complete a new feature and all tests pass and finally a *master* branch tracking formal product releases. The subsequent text will refer to these terms often.

6.1 AAMA Wireframes

6.1.1 Create AS IS representation

1. Import code - Before the product manager and software architect can create an accurate AS IS model, they import the product's source code. AAMA then uses contextual programming language information to generate code component building blocks for the model. In the example, the tool recognizes the MVC structure and automatically creates different components for models, views, controllers and makes initial assumptions about the relation between them. However, this generated architectural model view is not entirely correct and needs manual adjustment.

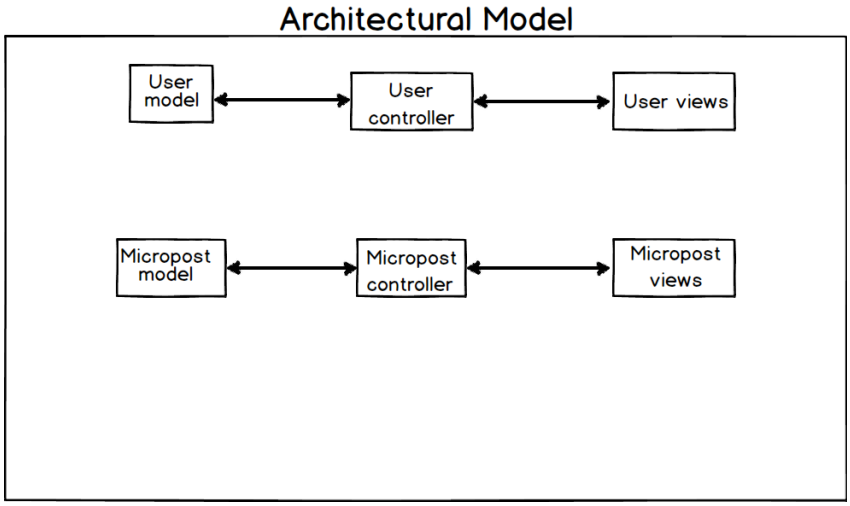


Figure 6.1: Twitter example 1: models, views and controllers are automatically generated

2. Model Construction - The product manager and software architect collaborate to construct a correct model. They group the generated components and connect them to one another when relevant to create an accurate AS IS representation. In the example, the stakeholders create two groups and connect the micropost controller to the user model, because the controller creates microposts through the user.

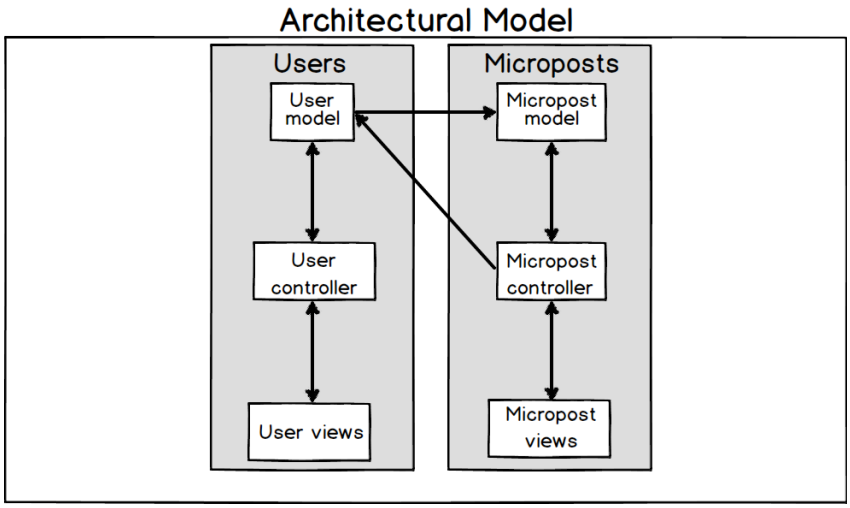


Figure 6.2: Twitter example 2: after constructing the model

3. Connect requirements list - Next, the product manager and architect import the historical list of requirements from their issue tracker. They connect each requirement to the relevant modules, sub modules and code components. Once complete, two views are available: a model with multiple levels of detail and a requirements table with information on which components each requirement is connected to. We see that our example application thus far is straightforward. Requirement POST-3, however, is peculiar. While a product manager might expect that viewing a micropost is part of its respective controller, it is actually called on the show method of the user controller. This is an example of how a product owner can achieve a deeper understanding of the application logic thanks to AAMA.

Requirements

Requirement	ID	Module(s)	Sub module(s)	code component(s)
User can login	USER-1	User	USER-AUTH	user controller, user model
User can delete his account	USER-2	User	USER-MGMT	user controller
User can update his account information	USER-3	User	USER-MGMT	user controller
User remains logged in when he closes browser	USER-4	User	USER-MGMT	user model
User can make microposts	POST-1	Micropost	POST-MGMT	micropost controller
User cannot make microposts for other users	POST-2	Micropost	POST-MGMT	micropost controller
User can view microposts	POST-3	User	USER-SHOW	users controller, user model, micropost model
User can delete microposts	POST-4	Micropost	POST-MGMT	micropost controller

Figure 6.3: Twitter example 3: Connecting historic list of requirements

6.1.2 Model Details

4. Inspecting a component - Now that all components are connected, users can inspect model components and see further details on a lower product level. In the example, our user is viewing details of the microposts controller which has three methods (create, destroy and correct user), some of which interface with other components on the same level as the microposts controller. Right next to the architectural model is a list of requirements that relate to the selected component, corresponding with the three requirements that relate to the microposts controller in Figure 6.3.

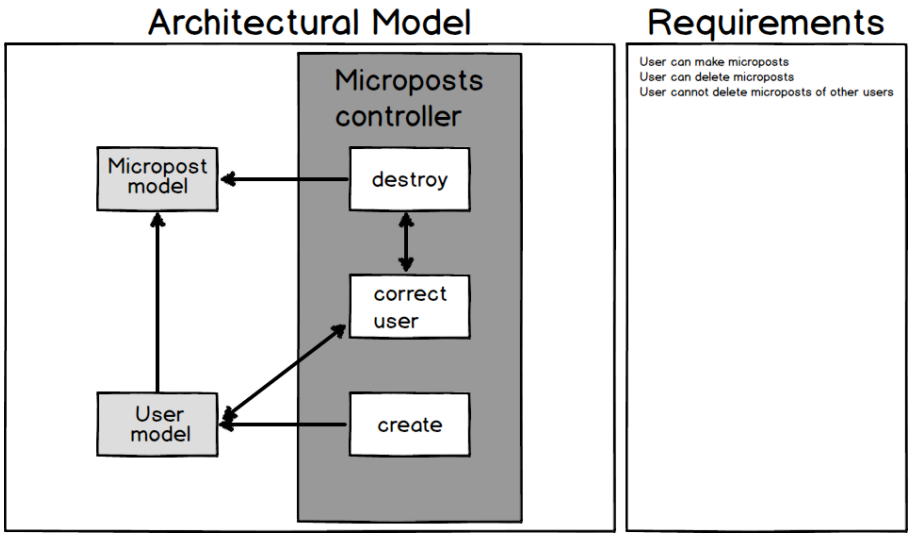


Figure 6.4: Twitter example 4: Inspecting the microposts controller

5. Inspecting a requirement - In a similar manner, users can select requirements from one of the requirements lists to see their corresponding model components and other related requirements. In the example, we see that the requirement *"User cannot delete requirements of other users"* primarily concerns the *correct user* and *destroy* methods of the microposts controller and relates to the two other requirements we saw earlier.

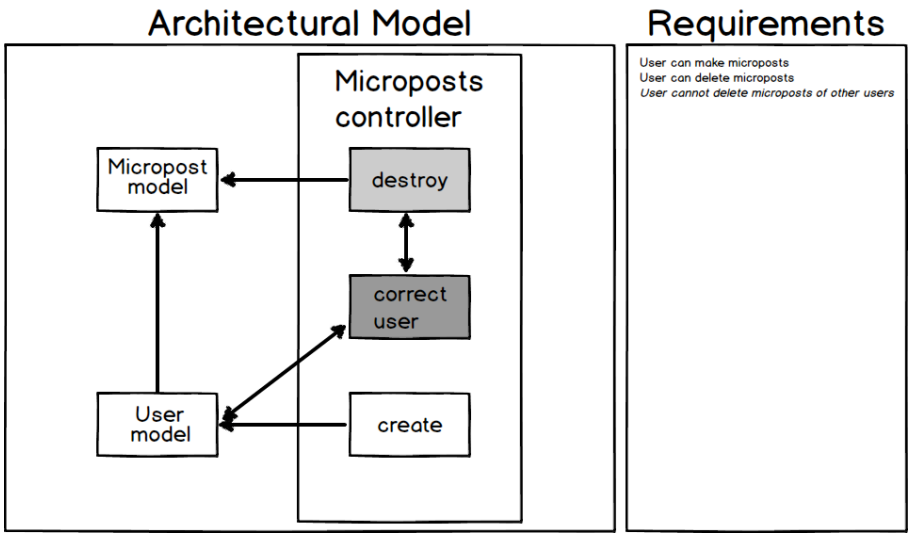


Figure 6.5: Twitter example 5: Inspecting the microposts controller

6.1.3 TO BE Representation

6. Import new requirements - Whenever the product manager processes one or more new requirements in the issue tracker, AAMA automatically imports them. The example show one new requirement: *a user can follow other users.*

Requirements

Requirement	ID	Module(s)	Sub module(s)	code component(s)
User can login	USER-1	User	USER-AUTH	user controller, user model
User can delete his account	USER-2	User	USER-MGMT	user controller
User can update his account information	USER-3	User	USER-MGMT	user controller
User remains logged in when he closes browser	USER-4	User	USER-MGMT	user model
User can make microposts	POST-1	Micropost	POST-MGMT	micropost controller
User cannot make microposts for other users	POST-2	Micropost	POST-MGMT	micropost controller
User can view microposts	POST-3	User	USER-SHOW	users controller, user model
User can delete microposts	POST-4	Micropost	POST-MGMT	micropost controller
User can follow other users	RELA-1			

Figure 6.6: Twitter example 6: Importing new requirements

7. Connect new requirements - For each new requirement, the product manager enters into the requirements table which modules and code components he expects the requirement to relate to. Next, he draws connections between the modules on the architectural model as a specification for the development team. The end-result is a new version of the architectural model, with the new requirement included. The current architectural model now represents a TO BE situation of the product, while the contemporary AS IS model is no longer shown to users in the application. In the example, we see that the product manager thinks that following a user is called by the user controller to the relationship controller and subsequently the relationship model.

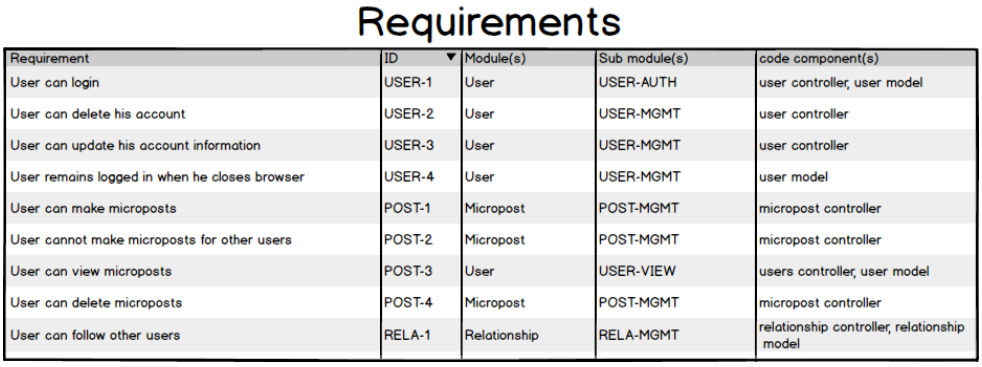


Figure 6.7: Twitter example 7: Connecting new requirement

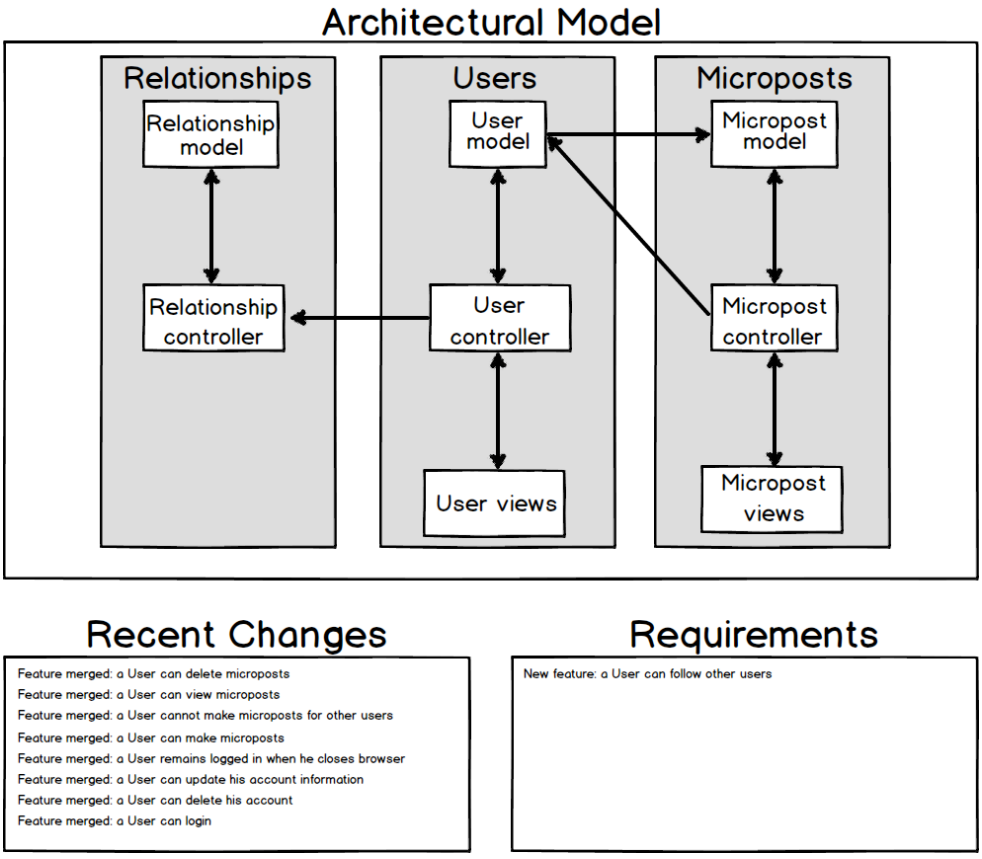


Figure 6.8: Twitter example 7: Expanding model based on requirement

6.1.4 Validate, Adjust and Review Model

8. Consulting the model - When the development team starts a new requirement, the software architect consults the current model for implementation guidance. Every new requirement starts on a branch, which has the requirement ID in its name.

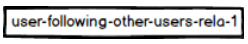


Figure 6.9: Twitter example 8: branch name for RELA-1

9. Code analysis, model adjustment and verification notification - When a new requirement branch is merged into master, AAMA analyzes the code and compares it to the current model as imagined by the product manager. AAMA automatically adjusts the architectural model when the implementation diverges from the prescribed model. In the example, we see that the application adds two new connections: user views call the relationship controller and the relationship model has a belongs_to relationship with the user model. After code analysis and automatic model adjustment, the system displays to any visitor that the software architect has pending adjustments he needs to review.

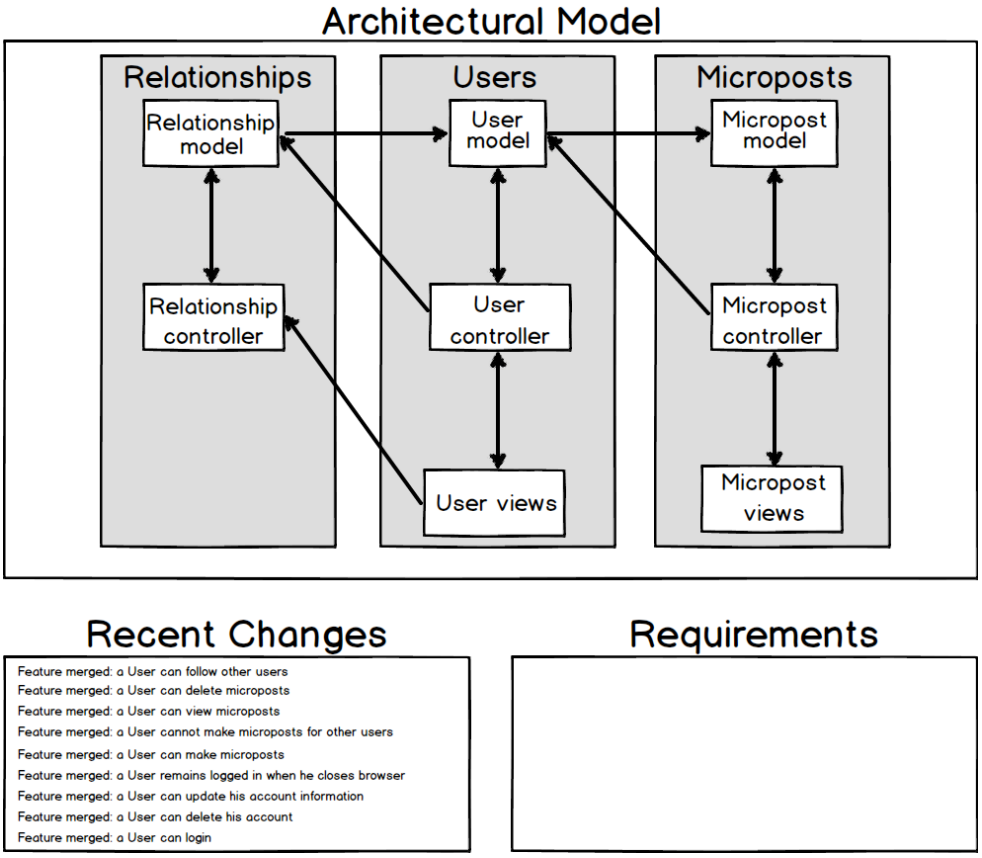


Figure 6.10: Twitter example 9: New view after code analysis

10. Adjustment review - When AAMA adjusts the model, a notification prompts the software architect to review the proposed changes. If necessary, he further adjusts the model to create a fully accurate current model representation. In our example we see that the user controller never actually calls the relationship model, prompting the software architect to remove the connection from both the model and the requirements list.

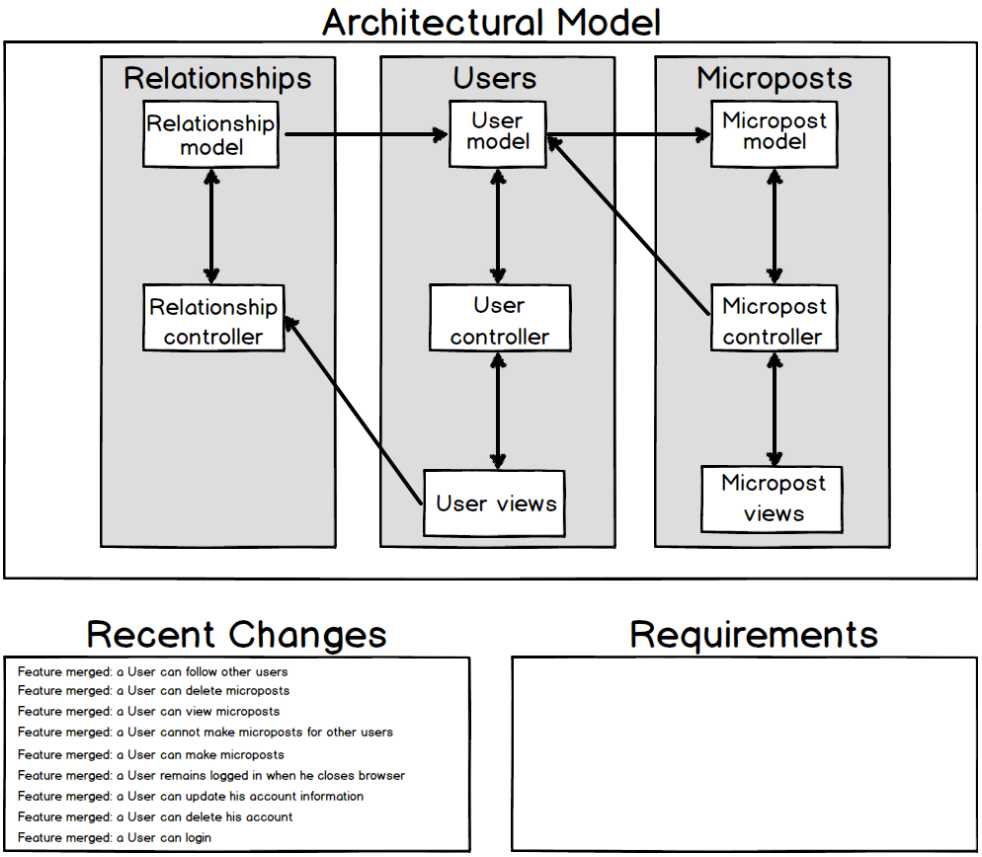


Figure 6.11: Twitter example 10: model after software architect makes changes

Requirements

Requirement	ID	Module(s)	Sub module(s)	code component(s)
User can login	USER-1	User	USER-AUTH	user controller, user model
User can delete his account	USER-2	User	USER-MGMT	user controller
User can update his account information	USER-3	User	USER-MGMT	user controller
User remains logged in when he closes browser	USER-4	User	USER-MGMT	user model
User can make microposts	POST-1	Micropost	POST-MGMT	micropost controller
User cannot make microposts for other users	POST-2	Micropost	POST-MGMT	micropost controller
User can view microposts	POST-3	User	USER-VIEW	users controller, user model
User can delete microposts	POST-4	Micropost	POST-MGMT	micropost controller
User can follow other users	RELA-1	Relationship	RELA-MGMT	relationship controller, relationship model, user views

Figure 6.12: Twitter example 10: requirements list after software architect makes changes

11. Historical perspective - Whenever a new feature branch merges into the development branch, AAMA creates a new model. Using the different branches as a timeline, AAMA creates a historical view of the architecture that allows stakeholders to walk through the application's creation over time. This form of traceability is beneficial for reconstructing why specific decisions were made in the past and for educating new architectural employees. In the example, we see a list of branches that users can select to travel to a previous state.

6.1.5 Feature Roadmap

The following features describe advanced functionality and challenges that are out of scope for a minimum viable version of AAMA. If this first version of AAMA is a success, we plan to further extend it by iteratively incorporating more and more of these features.

12. Missing requirements insights - AAMA analyzes all connections between architectural components and requirements to identify specific components without any corresponding requirements. This allows the product manager to take advantage of undocumented functionality for marketing purposes or make informed decisions on pursuing new requirements that relate to existing functionality.

13. Critical components - AAMA again analyzes all connections between architectural components and requirements. This time, the goal is to identify components that connect with a large number of corresponding requirements or specific requirements that are particularly critical to product value. An example advantage is that the product manager gains a better understanding of components that justify investment to solve technical debt.

14. Filtering components - AAMA currently has one architectural model of the product that is identical to all stakeholders. By adding special filtering options, stakeholders can select only those sections relevant to the discussion at hand. The resulting model is similar to an incidental model, focusing discussants attention to only the relevant modeling sections.

15. Clustering requirements - AAMA clusters requirements by name to identify trends in the product's features and potential for synergistic advantages.

16. Quality requirements in models - Integrating quality attribute requirements in AAMA's architectural model adds another dimension to product manager and software architect information exchange.

6.1.6 AAMA Impact

In the previous section we noted that we aim to achieve continuous architectural model accuracy. The AAMA prototype has a highly iterative approach, with functionality 4 through 10 occurring constantly and in parallel with one another. As a consequence, the model is perpetually evolving towards its future state. While this means that the model is never *completely* accurate, the quick iterations and frequent verification by stakeholders prevent human error and enforce the architecture to become a living document. In turn, these measures ensure the model's timeliness and approximate accuracy over the product's entire lifetime, our primary objectives. Secondary, the historical perspective enables users to utilize the architecture for analytical and educational purposes.

These benefits first and foremost impact the software architect, whose work requires less effort when all stakeholders use accurate and up-to-date models. The product manager primarily uses AAMA for communicating new requirement specifications to the software architect. Furthermore, the direct feedback when AAMA adjusts his prescribed models educates him on the correct approach for future reference and ensures his technical product knowledge is correct and contemporary.

6.2 AAMA Evaluation - Research Approach

We evaluate our prototype in two stages: case studies and a survey. We conduct five semi-structured interviews following the case study method described in S. Jansen and Brinkkemper (2008). The five case study companies all originate from and are currently based in the Netherlands, but their size, age and target markets differentiate. Each case study consists of two interviews of approximately 45 minutes with a product manager and software architect, except for SPO 4 of which we only interviewed a product manager.

The objective of the evaluation is to measure how practitioners perceive AAMA and whether they are interested in applying AAMA in their day to day operations. We base the interview and survey questions on three concepts of the Unified Theory of Acceptance and Use of Technology (UTAUT) model:

1. Performance Expectancy
2. Effort Expectancy
3. Behavioral Intention

In 2003, Venkatesh et al. (2003) presents UTAUT based on eight prominent, earlier acceptance models. They formulate five empirically validated determinants to reliably predict technology usage. In our research, we drop the determinants *social influence* and *facilitating conditions* because their questions require a concrete implementation case instead of a conceptual prototype. While UTAUT is typically conducted by means of a survey, the exploratory nature of our research approach prohibits constructing a strong survey. The case studies function as a pilot for the survey. The results allow us to assess the initial prototype and formulate a more refined second version that offers more value to the relevant stakeholders. Moreover, we re-evaluate the relevance of our interview questions to construct a more relevant survey.

The interviews contain sections with unguided questions, guided questions and cross-validation topics. First, we introduce the researcher and research topic. Next, we evaluate the role of product management and architecture in the organization. Subsequently, we guide the respondent through our prototype and ask questions such as: *"Do you think AAMA is useful for requirements gathering?"* and *"Do you expect AAMA to contribute to information exchange effectivity?"*. In total, the research poses eight questions to determine respondent's perception of AAMA and their usage intention. Additionally, we include question that will (in)validate our earlier findings on the roles and main collaborative activities of product managers and software architects. Further research protocol details are available in Appendix B.

6.3 AAMA Evaluation - Case Study Results

Experience and Technical Competence - Respondents' years of experience in their respective role varies from 1 to 15 years, with an average of 7. Product managers and software architects estimate the product manager's technical competence close to one another. Averaged, the product managers give themselves a 3,6 on a scale from 1 to 7. Software architects give their product managers a 3.

Collaborative Activities - The intention of this question was to confirm whether or not product managers and software architects collaborate in the activities as identified in our initial case study. This second series of case studies confirm parts of our initial findings. Out of a total of 9 respondents, 7 indicate that requirements refinement is a collaborative activity and 5 conduct requirement prioritization together. However, just one product manager and one software architect indicate the architect provides input for requirements gathering. Finally, 4 respondents indicate that collaboration is important for product roadmapping. These results require a disclaimer: 4 out of 9 respondents also took part in our initial case study, reducing respondent diversity of these answers.

	PE1	PE2	PE3	PE4	EE1	EE2	EE3	BI
SPM 1	Yes	Yes	Yes	No	Yes	Yes	Yes	3
SPM 2	No	No	No	No	Yes	Yes	No	No
SPM 3	No	No	No	No	Yes	Yes	No	No
SPM 4	No	No	No	No	Yes	Yes	Yes	No
SPM 5	No	No	No	No	Yes	Yes	Yes	No
SA 1	No	No	No	No	Yes	Yes	Yes	No
SA 2	Yes	No	Yes	No	Yes	Yes	Yes	12
SA 3	Yes	No	No	No	Yes	Yes	No	No
SA 4	Yes	No	Yes	Yes	No	Yes	No	1
Yes	4	1	3	1	8	9	5	
No	5	8	6	8	1	0	4	

PE 1: I think AAMA is useful for my job
PE 2: Using AAMA enables me to accomplish my tasks more quickly
PE 3: Using AAMA increases my productivity
PE 4: If I use AAMA, I will increase my chances of getting a raise
EE 1: My interaction with AAMA would be clear and understandable
EE 2: It would be easy for me to become skillful at using AAMA
EE 3: I would find AAMA easy to use
BI: I intend to use AAMA in the next n months

Table 6.1: Raw data for UTAUT questions

Performance Expectancy (PE) - The first four UTAUT questions test whether respondents believe the prototype has a positive impact on their work. Unfortunately, respondents indicate that the initial version of our prototype provides insufficient benefits to justify expanding their processes. Of the 36 responses to 4 questions related to performance expectancy, 27 answers were negative. Nevertheless, 4 respondents did respond to the question *"Do you think AAMA is useful for your job?"* with yes. All respondents but one do not think that AAMA enables them to complete their day to day tasks more quickly. 3 respondents think that AAMA will increase their productivity, although 2 added the caveat that this is only applicable for large products. Merely one respondent thinks using AAMA will increase his chances of getting a raise.

Effort Expectancy (EE) - The next three UTAUT questions concern respondents' perception of the effort required of them to use the system. The results are contrary to those for expected performance. Of 27 responses, 22 were positive. 4 out of 5 negative responses concern finding AAMA easy to use. Aside from the final negative response, all other respondents believe that their interaction with a "real" version of AAMA would be clear and understandable. Every

respondent is convinced it would be easy for him/her to become skillful at using AAMA.

Behavioral Intention (BI) - UTAUT includes one question to assess whether respondents intend to use the prototype: "I intend to use AAMA in the next n months". Additionally, the form of this question provides an indication of the respondent's urgency to start using AAMA. However, just 2 respondents noted they would use AAMA within a short time frame: 1 and 3 months.

Additional requirements - Respondents gave varying answers when asked if they can come up with one additional requirement they believe would provide the most value to them. A frequent request was to show architectural model on a slightly higher abstraction layer, to reduce the effort required to create models. For 2 respondents, this is their most valuable future requirement. Clustering requirements and clustering code components provide the most value to 2 respondents each. The following three requirements are preferred by single respondents: (1) seeing requirements on flows between architectural components, (2) reports on requirements' test coverage and (3) the impact of new requirements on code *and* requirements, which incidentally requires both component and requirement clustering.

6.3.1 Preliminary Discussion

Our respondents' average years of experience indicates they are fairly experienced professionals in the product software industry. In general, product managers have sufficient technical competence to understand technical questions and issues, but not to solve them. In one specific case, the product manager intentionally pretends to be less technically competent than he is. He has a familiar motivation: getting out of the software architect's way as much as possible, to permit him to do his best work.

From previous literature and our initial case studies, we did not find any indication that product roadmapping is a collaborative activity. Yet, for many respondents it is. This was somewhat of a surprise. Although we recognize that software architects should contribute input during the process, we did not think that the product manager and software architect create roadmaps together. Upon closer inspection of the data, however, it appears that respondents that collaborate on product roadmapping without fault indicate they or their product managers have little technical competence. The product manager elicits technical knowledge from the software architect to supplement his knowledge of future technical opportunities.

Another result concerning collaborative activities surprises us: just 2 respondents claim to collaborate on requirements gathering. Based on careful analysis of the interviews, we believe this result to be negatively skewed. For the average product manager's frame of reference, the activity 'requirements gathering' stands for eliciting new requirements from external stakeholders. From the product manager's point of view, the software architect does not actively elicit requirements from external stakeholders and thus 'requirements gathering' is not a collaborative activity. From our point of view however, the software architect is an important *internal* stakeholder that contributes unique requirements. As the one responsible for maintaining the product's technical integrity, the software architect needs to collaborate with the product manager to select the most important technical issues to resolve. This realization prompts us to reformulate these questions for the survey into concrete, direct question instead of providing a multi-select list. For example 'does the software architect contribute requirements during requirements gathering?'

The UTAUT results entice some interesting discussion. Concerning performance expectancy, 3 out of 4 respondents who think AAMA is useful for their job were software architects. An indicator that AAMA's focus on architecture is too strong. Furthermore, we agree with respondent's belief that AAMA does not enable you to work more quickly. Indeed, the opposite is true. AAMA requires additional work from all stakeholders to generate meaningful contributions.

Additionally, the notion that using a new system leads to improved financial compensation is incomprehensible to all respondents except one. At least not within their organization. Perhaps this sentiment is caused by underlying cultural differences compared to the originating culture of UTAUT, the USA. This prompts us to remove this question from the final survey.

The overwhelmingly positive results concerning effort expectancy indicate that we made the right decision by keeping our approach to modeling and working with models as simple as possible, an effort we will maintain for the next iteration. However, the 4 out 5 negative responses for finding AAMA easy to use remains troubling. When asked, respondents state they believe the prototype requires extra attention in terms of usability design and polish to properly direct users to the right screens. A valid criticism which we address in V2 of the prototype.

6.3.2 General Feedback and Suggestions

We identify four major themes mentioned by respondents during the interviews: (1) product managers do not construct architectural models, (2) the prototype uses too much detail, (3) the technical challenge is too large and (4) knowing which components comprise a requirement and vice versa is valuable. This subsection discusses each theme in more detail.

Product Managers Create Architectural Models Although their reasons vary, every single respondent has reservations concerning product managers constructing architectural models. Product managers in particular believe that the average product manager does not want to create models. On top of that, respondents are especially resolute in their conviction that product managers *should* not want to formulate models in the first place. Our initial premise that "it won't hurt to try" and that product managers' attempts will improve over time fell to deaf ears. They argue that specifying 'how' to approach a technical problem impedes developers and software architects' freedom and creativity, ultimately leading to a weaker product.

> "Whenever I specify a requirement with too much detail, I can see that I pause the developer's brain because he figures the solution has already been thought up. Because I am weaker on a technical level, the end product is worse than if I hadn't specified anything at all."

Prototype Detail The majority of respondents, 6 out of 9, think the level of detail that AAMA's architectural models go into is too high. Product managers do not grasp the functional intention from these models and software architects do not require this much architecture detail for their daily decisions. Moreover, respondents raise some practical issues. One respondent's organization has multiple very large and complex requirements that would require an entire page to connect all code components. Another SPO began an effort to model their entire product two and half years ago, thus far they have models for approximately 25%.

Technical Challenges When the prototype claimed to utilize contextual knowledge to automatically create and modify architectural models, 6 respondents immediately voiced their skepticism. They do not question the feasibility if you constrain yourself: a simple product, a limited set of programming languages or an organization that strongly adheres to conventions. However, they do doubt the applicability to their (decade old) products with more than two million lines of code among three different programming languages and self-written framework with esoteric conventions.

Requirements and Components Despite these challenges, 6 respondents emphasize there is considerable value in being able to view the connections between requirements and code components. If this can be attained (partially) automatically, the value grows even further. Especially for those difficult, decade old products which have considerably more technical debt than younger, simpler products. The exact value respondents derive from this information and

how they would like to further operationalize the data varies. One product manager believes the current connections enables clear reasoning about their interrelationships, another respondent would like to see how different requirements impact other requirements and yet another sees value in knowing how new requirements impact specific code components.

Chapter 7

Improving AAMA

The previous chapter concluded that respondents believe the initial AAMA version provides insufficient value and that multiple aspects of the prototype take an incorrect approach. In the next section, we introduce improvements points based on the UTAUT results and general feedback in the previous chapter. Subsequently, we present wireframes of the second prototype version that integrates the improvement points. In the final section, we present the results of an online survey to evaluate the prototype with more practitioners.

7.1 Improvement Points

Prescribing architecture - From the case studies, it is apparent that product managers do not want to and should not create architectural models. Instead, the product manager has the option to assign the requirement to the relevant module.

Architectural freedom - The initial version prescribe a deliberate amount of detail a software architect is expected to go into when modeling the product. While the majority of respondents considered this to be too fine-grained, they state that for some product areas this level of detail *is* relevant. As a solution, we permit the software architect the freedom to choose the relevant detail amount for every requirement.

Architecture reconstruction - When a requirement is ready to be merge into the 'master' branch, the software architect creates a model of strictly the relevant components with the amount of detail he deems sufficient. The minimum requirement is on the module level, which AAMA utilizes to construct a high level architecture view. Once completed, the software architect merges the requirement and AAMA includes the components in the requirements list.

Product manager notification - Upon merge, the product manager is notified of the completion of a requirement via email and is shown the accompanying architectural model and code components. He can comment on the model if he has any questions.

Model database - Over time, the architectural model collection grows into a comprehensive database that the product manager and software architect consult for technical discussions when necessary.

Presentation - Finally, respondents react negatively to the guiding steps and role division prescribed in the prototype. Instead of focusing on AAMA's potentially beneficial features, the approach points their attention to drawing comparisons to their own development process, which invariably did not align. We present the second version of the tool prototype feature by feature to grant respondents the chance to evaluate each feature's individual merits. Furthermore, explanatory text layers on top of the application windows replaces long text between each step in an effort to promote the application's flow.

7.2 Wireframes

1. AAMA Landing - The prototype starts with an overview of the AAMA dashboard when the developers have already implemented basic functionality for users and microposts. Unlike the previous prototype, we presume there already is a limited amount of data available on specific requirements. On the dashboard, the user sees (1) a very high level architectural view of the application, (2) a list of current requirements, (3) an overview of the recent changes and (4) links to other application screens. When the user clicks next slide, the prototype instructs him to click on requirement POST-4.

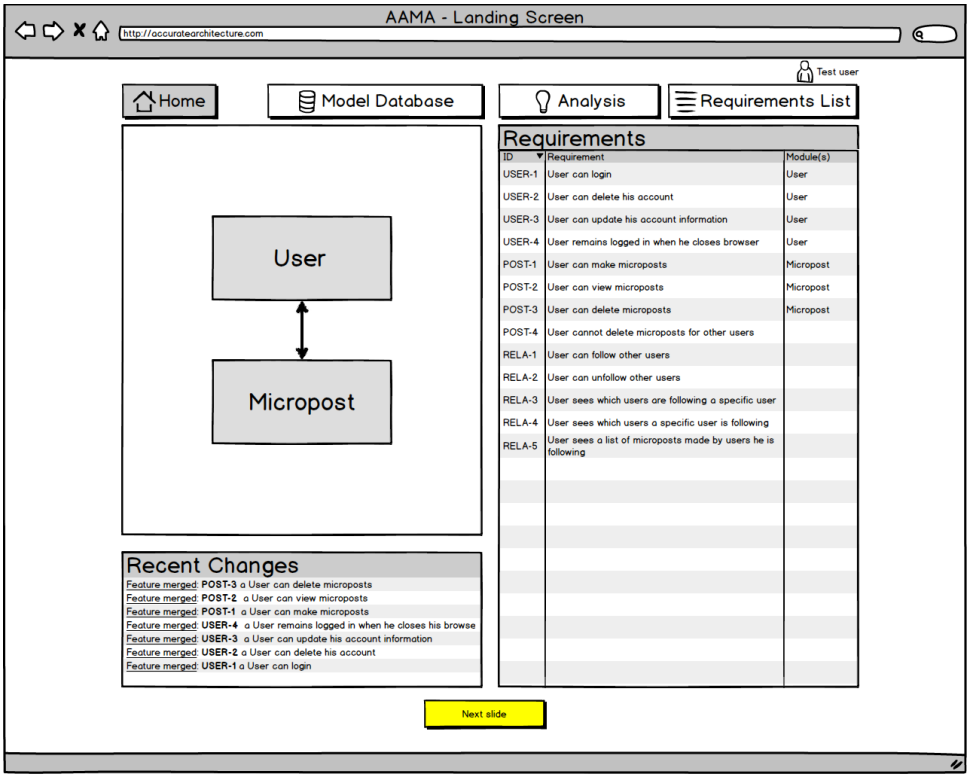


Figure 7.1: Twitter example 2.1: AAMA landing screen

2a. Create Model - The software architect consults the requirement specification for POST-4 and implements it. Next, he uses AAMA's modeler to create a model representation of his solution. Depending on the complexity and relevance of the requirement, the software architect decides to document the requirement on a lower or higher detail level.

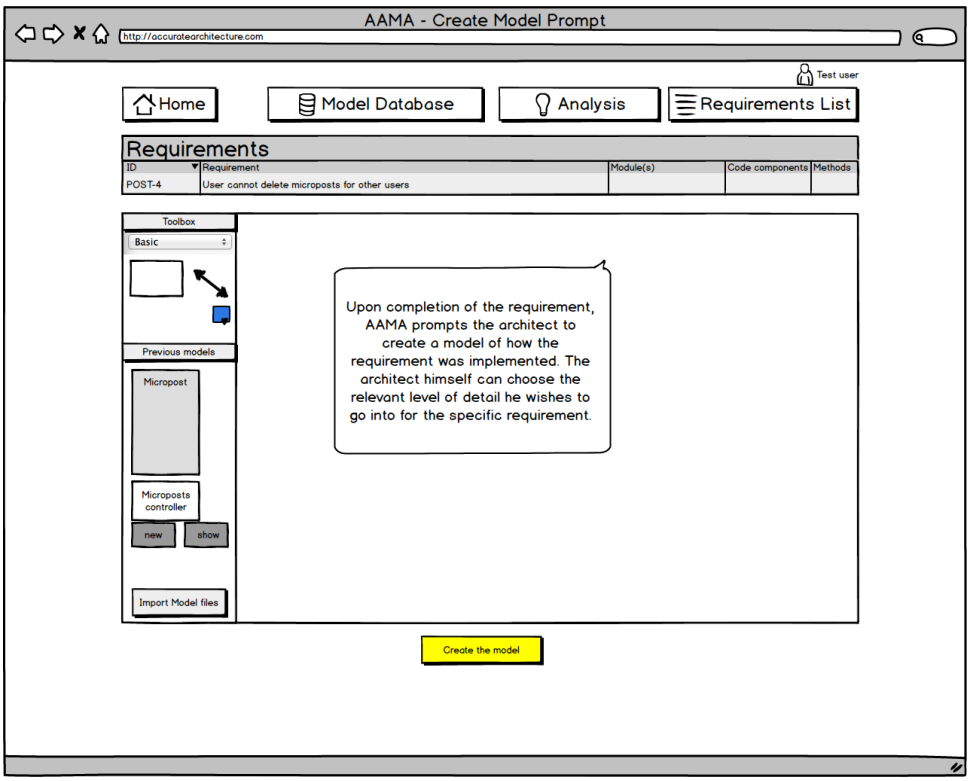


Figure 7.2: Twitter example 2.2a: AAMA modeler

2b. Create Model - For requirement POST-4, the software architect has chosen to go into considerable detail. He documents the modules, all code components and externally exposed methods. For the model, however, he decides to add the internal method *correct_user* as well.

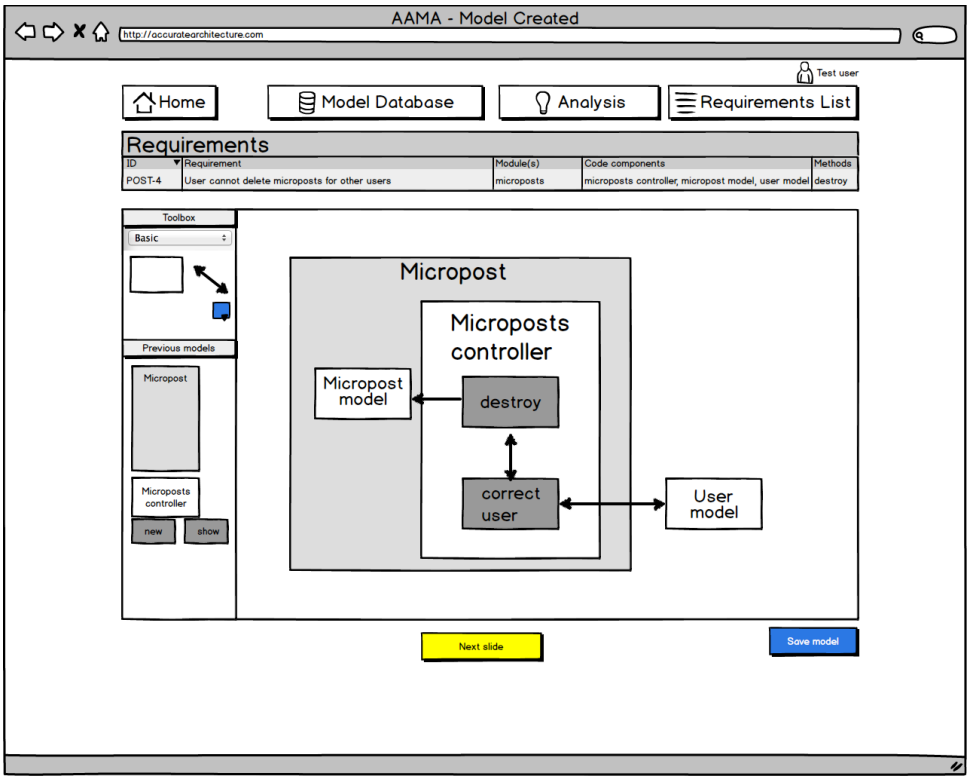


Figure 7.3: Twitter example 2.2b: model created

3. Stakeholder Notification - Relevant stakeholders such as product managers, developers and other software architects subscribe to a project to receive notifications of changes. The moment the software architect completes his model, AAMA notifies all subscribed stakeholders by sending the email below. The email includes an image of the relevant architectural model and a link to the detail view on the website.

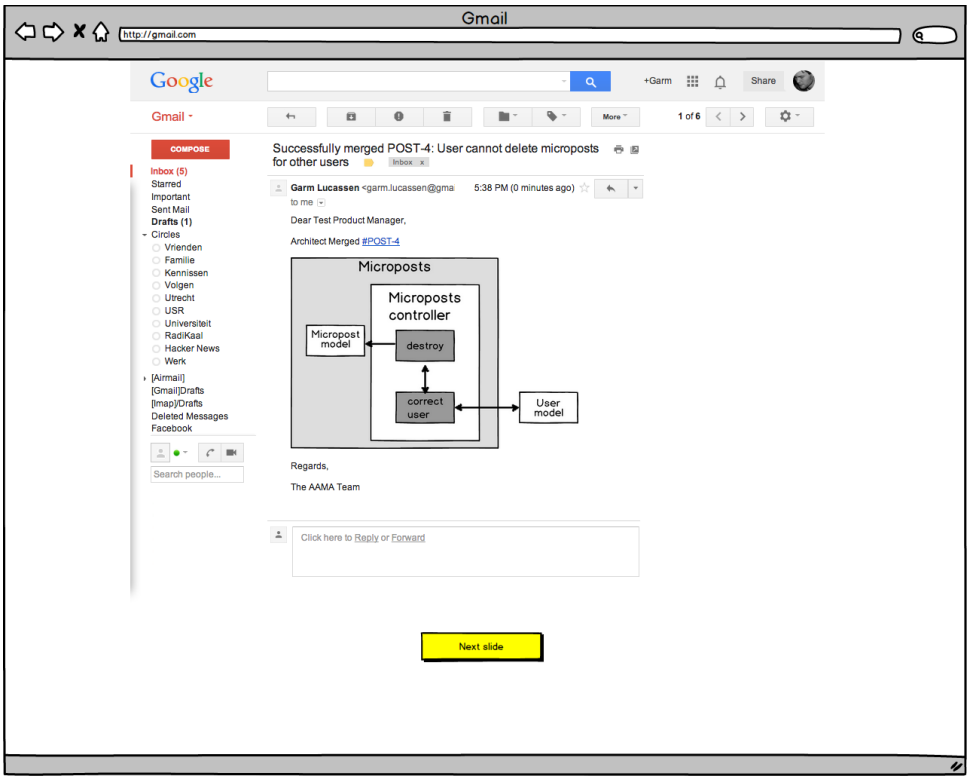
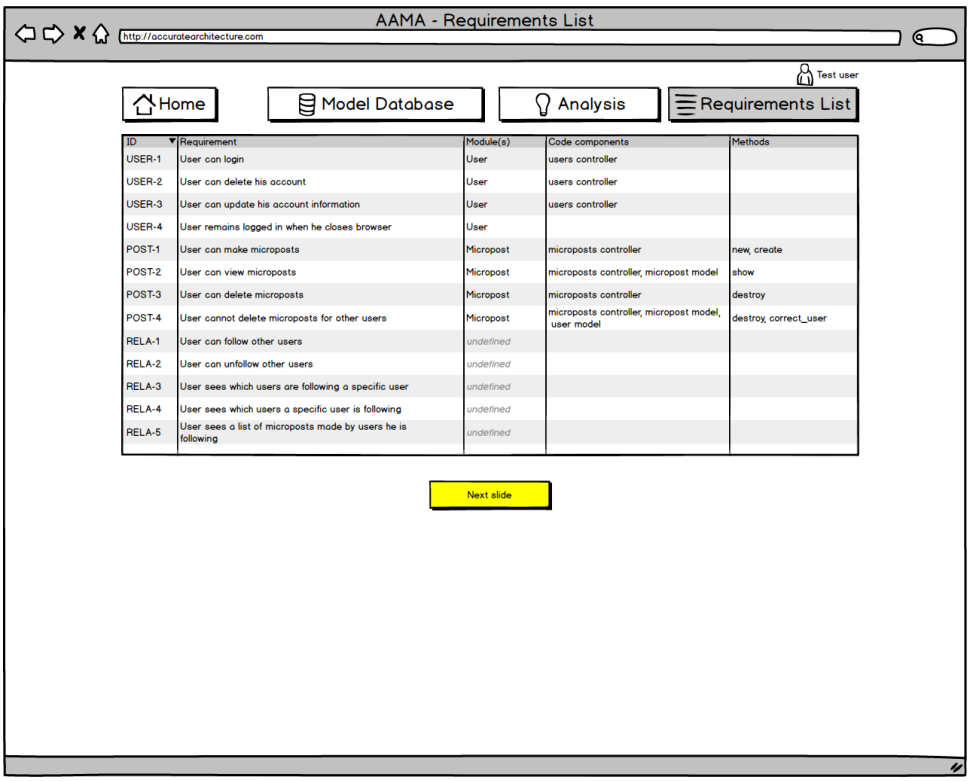


Figure 7.4: Twitter example 2.3: relevant stakeholders receive an email with the model

4. Requirements List - AAMA presents users with a list of all requirements and the details as specified by the software architect. Requirements with an *undefined* module are unimplemented. In the example we can see that the software architect did not think it was necessary to specify much detail for basic user requirements. The user can click any of the requirements in this list to view its architectural model.



ID	Requirement	Module(s)	Code components	Methods
USER-1	User can login	User	users controller	
USER-2	User can delete his account	User	users controller	
USER-3	User can update his account information	User	users controller	
USER-4	User remains logged in when he closes browser	User		
POST-1	User can make microposts	Micropost	microposts controller	new, create
POST-2	User can view microposts	Micropost	microposts controller, micropost model	show
POST-3	User can delete microposts	Micropost	microposts controller	destroy
POST-4	User cannot delete microposts for other users	Micropost	microposts controller, micropost model, user model	destroy, correct_user
RELA-1	User can follow other users	*undefined*		
RELA-2	User can unfollow other users	*undefined*		
RELA-3	User sees which users are following a specific user	*undefined*		
RELA-4	User sees which users a specific user is following	*undefined*		
RELA-5	User sees a list of microposts made by users he is following	*undefined*		

Figure 7.5: Twitter example 2.4: requirements list

5. Model Database - Over time, creating a model for each requirement generates a large requirement database. AAMA provides users a quick overview of these models (and in turn their product) by grouping all models on the model database page. In the example three things stand out. (1) the software architect has decided not to create any models for the user module. Likely because it is not interesting enough to justify the work necessary. (2) The user can hide irrelevant models for his specific situation. (3) The detail level varies per requirement.

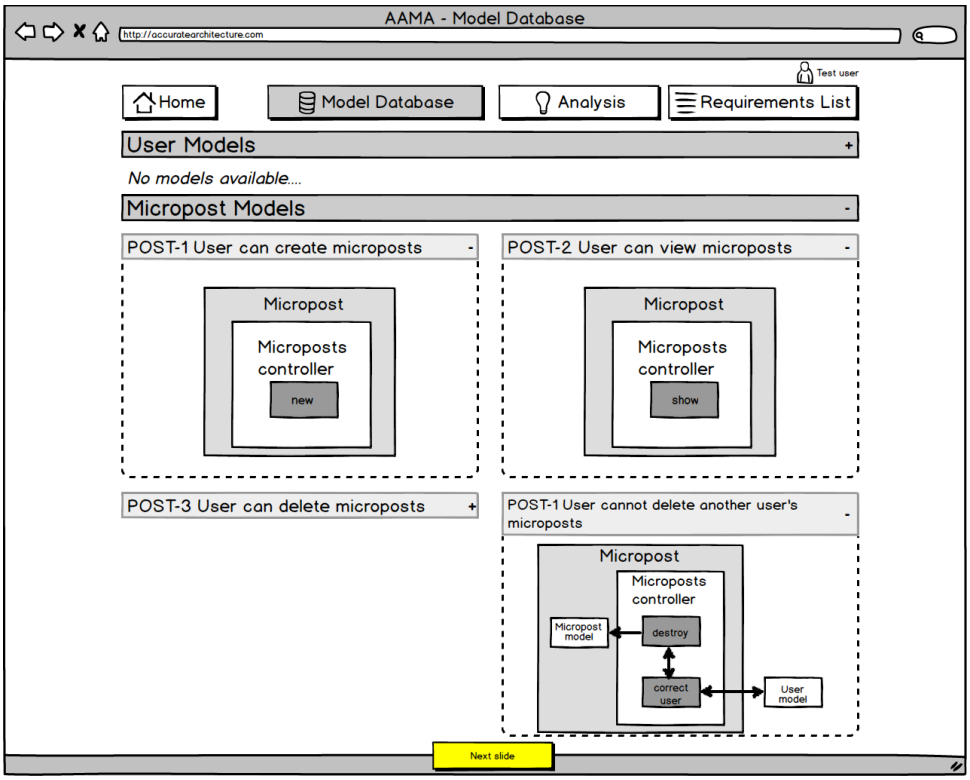


Figure 7.6: Twitter example 2.5: model database

6a. Landing Model Zoom - Now that we have seen the model database, we are ready to return to the landing screen and take a closer look at the model on the left side. AAMA automatically assembles this model by combining all manually created models and requirement details. Here we see the highest level representation consisting of just a User and Micropost model. Because the architect has created more models for Micropost, we click it to view more details.

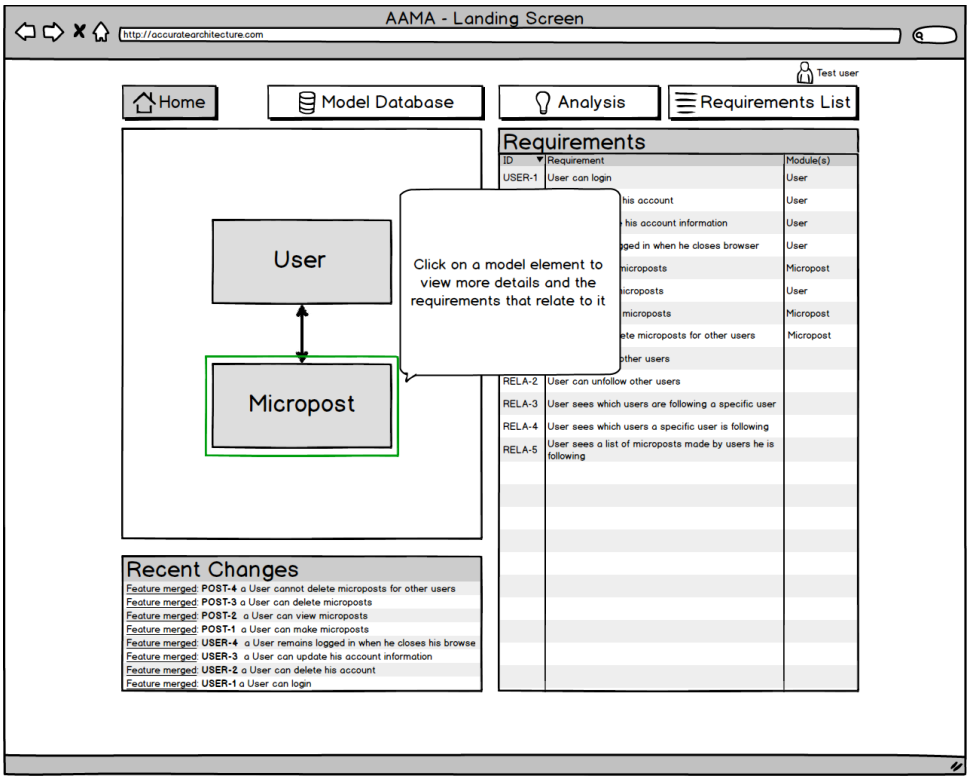


Figure 7.7: Twitter example 2.6a: zooming in on microposts

6b. Landing Model Zoom - Zoomed into the next level, the Micropost module displays its respective model and controller code components as defined in the requirements list and accompanying models. Additionally, the requirements list on the right side is reduced to only those requirements that relate to the components in the model.

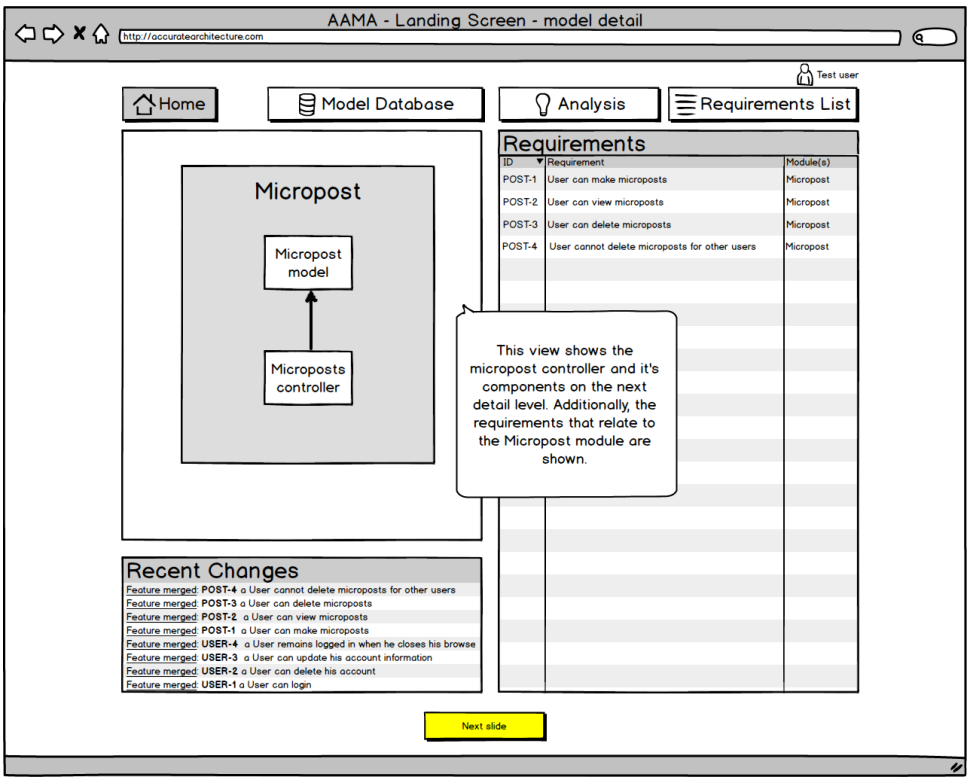


Figure 7.8: Twitter example 2.6b: zoomed micropost view

7. Analysis - The final screen consists of three separate functionalities. (1) A more detailed overview of the automatically assembled model. Depending on the product's complexity, AAMA automatically starts at a higher abstraction level to ensure that the entire model fits on the page. (2) Requirements clustering based on their ID, description and component information. For larger number of requirements, automatically abstracts and zooms requirement groups in a similar manner to the assembled model. (3) The critical components feature counts all component occurrences in the requirements list and sorts them by importance.

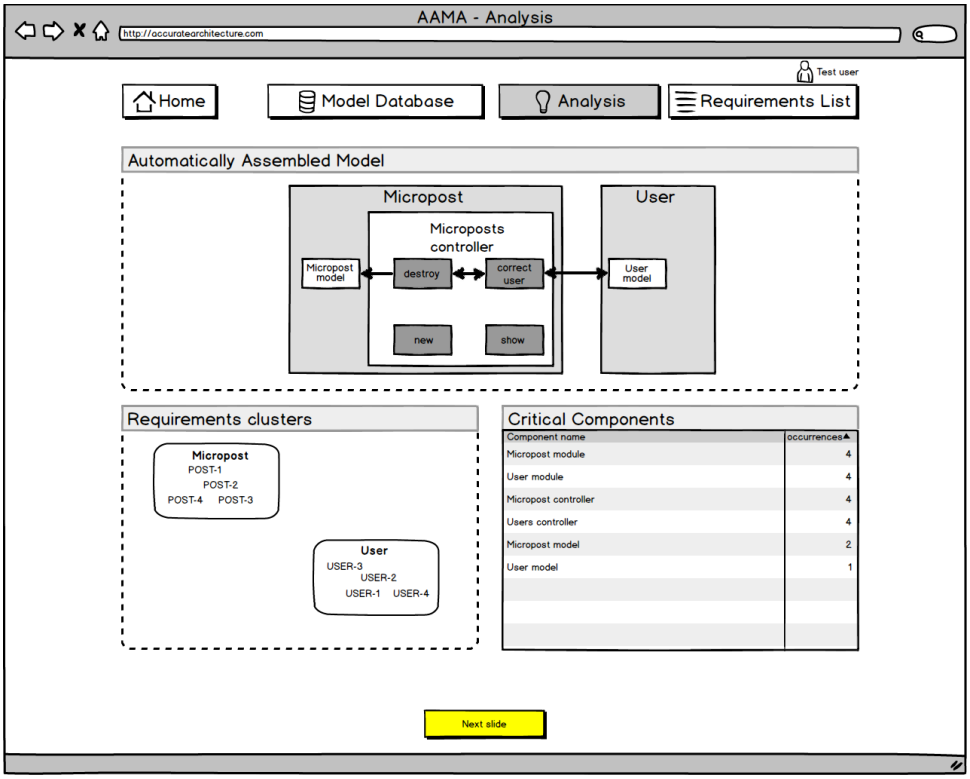


Figure 7.9: Twitter example 2.7: analyzing your product

7.3 AAMA Evaluation II - Survey

The survey has the same objective as the AAMA case study evaluation: to measure how practitioners perceive AAMA and whether they are interested in applying AAMA in their day to day operations. We expect prototype version II to elicit more positive responses from both product managers and software architects. To confirm this expectation, we adapted our case study questions to a survey. As mentioned in the case study results, we exclude whether respondents think that using AAMA increases their chances of getting a raise on cultural grounds. Furthermore, the survey includes three extra questions: an open question into whether the respondent thinks AAMA solves a problem for him and two yes/no questions to determine whether requirements clustering and critical components is valuable to respondents. In total, the survey consists of 3 context questions, 13 closed questions and 7 open questions. The entire list of questions is available in Appendix C.

7.3.1 Survey Results

Unfortunately, the limited number of respondents prohibits us to draw definite conclusions from the survey data. Despite distributing our survey among 94 product managers that previously participated in a software product management course for professionals, just 15 product managers responded to our request. We expected reaching software architects to be particularly challenging because Utrecht University has no database of software architects at SPOs. In an attempt to overcome this hurdle, the invitation for product managers included a request to distribute the survey among software architects in their organization. Although several product managers wrote back that they forwarded our request to one or more software architects, in the end only 1 successfully completed the survey. In a final attempt to increase the number of responses, several LinkedIn requests were posted to large, international groups with a focus on software product management or architecture. Unfortunately, this did not lead to any new responses.

It is easy to attribute this disappointing lack of responses to an external factor: many people are on holiday during the summer period and the remaining employees assume they do not have the time to fill out a survey. Until you take a closer look at the number of clicks for each survey. Approximately 100 product managers and 50 software architects visited the survey. A conversion rate of 15% for product managers is acceptable, but 2% for software architects is not. Moreover, it indicates that we did a poor job constructing the survey and communicating its importance. For example: asking respondents to download a file, switch between applications and return to the survey upon completion probably removes a significant number of respondents from the funnel. For future surveys with a survey, we intend to mitigate these issues with three adjustments: (1) a browser-integrated experience means the respondent doesn't need to leave his browser to fill out the survey, (2) a shorter survey to reduce the chance that respondents immediately close the page and (3) a step by step presentation, which immediately asks respondents to evaluate a specific screen after it is presented to him.

Due to an inexplicable technical issue the surveys for product managers did not include two questions: (1) *"I (will - might - will not) use AAMA when it is available"* and (2) *"I would find AAMA easy to use"*. The second discrepancy was caught by the researcher after the 7th submission, so the remaining 8 did answer the second question. Despite the small number of responses and erroneous questions, we discuss our results in the subsequent paragraphs.

	XP	TC	RG	RP	RR	PR
SPM 1	5	3	No	No	Yes	Yes
SPM 2	3	5	Yes	No	No	No
SPM 3	1	3	Yes	Yes	Yes	Yes
SPM 4	1	2	No	No	Yes	Yes
SPM 5	2	7	Yes	Yes	Yes	Yes
SPM 6	3	5	Yes	No	Yes	No
SPM 7	5	2	No	No	No	No
SPM 8	4	4	No	Yes	Yes	Yes
SPM 9	5	5	Yes	Yes	Yes	Yes
SPM 10	1	4	Yes	No	Yes	No
SPM 11	8	4	Yes	Yes	Yes	Yes
SPM 12	4	6	Yes	No	Yes	No
SPM 13	2	3	Yes	Yes	Yes	Yes
SPM 14	4	5	Yes	Yes	Yes	No
SPM 15	1	5	No	No	No	No
AVG	3	4				
Yes			10	7	12	8
No			5	8	3	7
SA 1	4	2	No	No	Yes	Yes

XP: Experience TC: Technical Competence
RG: Requirements Gathering RP: Requirements Prioritization
RR: Requirements Refinement PR: Product Roadmapping

Table 7.1: Raw data on respondents' experience, technical competence and collaborative activities

Experience and Technical Competence - Respondents' experience as a product manager ranged from 1 to 8 years, with an average of 3. On average, these product managers give their technical competence a 4 out of 7 in comparison to their software architects. Due to the small number of responses from software architects it is impossible to make comparisons between the estimations of product managers and software architects.

Collaborative Activities - Exactly like in the previous case study, the intention of this question was to confirm whether or not product managers and software architects collaborate in the activities as identified in our initial case study. In Table 7.1, we see that 10 product managers indicate they collaborate on requirements gathering versus 5 who do not. 7 product managers collaboratively prioritize requirements, while 8 do not. For product roadmapping, these numbers are the other way around. 12 product managers refine requirements together.

	PE 1	PE 2	PE 3	EE 1	EE 2	EE 3	BI
SPM 1	No	No	No	No	No		N/a
SPM 2	No	No	No	Yes	Yes		N/a
SPM 3	No	No	No	Yes	Yes		N/a
SPM 4	Yes	No	No	No	No		N/a
SPM 5	Yes	No	No	Yes	Yes		N/a
SPM 6	No	No	No	No	No		N/a
SPM 7	No	No	No	No	No		N/a
SPM 8	No	No	No	Yes	No	No	N/a
SPM 9	No	No	No	No	Yes	Yes	N/a
SPM 10	Yes	No	Yes	No	No	Yes	N/a
SPM 11	No	No	No	No	No	No	N/a
SPM 12	Yes	No	No	Yes	Yes	Yes	12
SPM 13	Yes	No	Yes	Yes	Yes	Yes	N/a
SPM 14	Yes	No	No	No	No	No	N/a
SPM 15	No	No	No	Yes	Yes	Yes	N/a
Yes	6	0	2	7	7	5	
No	9	15	13	8	8	3	
SA 1	No	No	No	Yes	Yes	Yes	N/a

PE 1: I think AAMA is useful for my job
PE 2: Using AAMA enables me to accomplish my tasks more quickly
PE 3: Using AAMA increases my productivity
EE 1: My interaction with AAMA would be clear and understandable
EE 2: It would be easy for me to become skillful at using AAMA
EE 3: I would find AAMA easy to use
BI : I intend to use AAMA in the next n months

Table 7.2: Raw data for UTAUT questions

Performance Expectancy - Again, the first three UTAUT questions test whether respondents believe the prototype has a positive impact on their work. Of the 45 responses to 3 questions related to performance expectancy, 37 were negative. In particular, the proposition *"Using AAMA enables me to accomplish my tasks more quickly"* only received negative responses, while *"Using AAMA increases my productivity"* garnered just 2 positive responses. Nevertheless, 6 product managers do agree with the proposition *"AAMA is useful for my job"*. Yet, when it comes to **behavioral intention**, just 1 product manager would start using AAMA within a year.

Effort Expectancy - Again, these three UTAUT questions concern respondents' perception of the effort required of them to use the system. Product managers expect more effort is required for V2 than V1. Only 7 out of 15 product managers believe their *"interaction with AAMA would be clear and understandable"* and it *"would be easy for me to become skillful at using AAMA"*. 5 out of 8 do find AAMA easy to use.

	PS	IX	RC	CC
SPM 1	No	No	Yes	No
SPM 2	No	No	No	No
SPM 3	No	Yes	Yes	Yes
SPM 4	Yes	Yes	Yes	Yes
SPM 5	No	No	Yes	Yes
SPM 6	No	No	No	No
SPM 7	No	No	Yes	No
SPM 8	No	No	No	No
SPM 9	No	Yes	Yes	Yes
SPM 10	Yes		No	Yes
SPM 11	Yes	Yes	Yes	No
SPM 12	Yes	Yes	Yes	Yes
SPM 13	Yes	Yes	Yes	Yes
SPM 14	Yes	Yes	No	Yes
SPM 15	Yes	Yes	Yes	Yes
Yes	7	8	10	9
No	8	6	5	6
SA 1	No	No	Yes	Yes

PS: Do you think AAMA will solve a problem for you?
IX: Do you think AAMA will improve the information exchange between SPM and SA?
RC: Is Requirements clustering valuable to you?
CC: Is critical components valuable to you?

Table 7.3: Raw data for problem solving, information exchange and product analysis questions

Problem Solving, Information Exchange and Product Analysis - The survey contains two questions to explore whether respondents think AAMA can produce the benefits it was created for. 7 product managers think AAMA will solve a problem for them. Without assistance, however, the majority of these product managers have difficulty articulating what the exact benefit is that AAMA could provide. Only 1 product manager has a specific use case that he could use AAMA for right now. Of the 8 negative product managers, one provided a particularly compelling reason why: "As a product manager the advantages are limited, because it doesn't support my core activities: roadmapping, prioritizing the backlog, writing user stories and creating visuals for requirements."

In total, 8 product manager believe AAMA has the capability to improve the information exchange with the software architect. Coincidentally, they correspond to those that think AAMA solves a problem for them. One product manager did not provide an actual answer to the question, resulting in 6 remaining, negative responses. The survey also included three questions to assess whether product managers believe the two types of analysis features offer value. Out of 15 responses, requirements clustering and critical components list are valuable to 10 and 9 product managers, respectively.

General Feedback - There was little cohesion between respondents' feedback. Instead, we shortly discuss two issues that we think are particularly applicable: (1) **Tool integration -** one respondent mentions that all organizations strive to have as little supporting applications as possible. AAMA currently is a stand-alone application, which we might have to re-imagine to integrate with companies' current development practices. (2) **Value for software architects -** the strong focus on architecture might reduce the relevance to product managers as exemplified by the quote above. A more prominent position for analysis features brings more balance.

7.3.2 Preliminary Discussion

In comparison to the previous case study, these respondents have substantially less experience but estimate to have a higher technical competence level on average. We can therefore presume that respondents have sufficient technical knowledge to comprehend issues that arise. This time, however, there appears to be no correlation between respondents' technical competence and whether they collaborate on product roadmapping or not.

Although we again asked respondents to state which activities they collaborate on, the way we presented the questions was different. Previously we asked respondents whether they collaborate on a specific activity, without clarifying what we think that activity entails. For the survey, we gave respondents a descriptive collaborative activity and asked them whether or not it applies to their organization. For example: 'Do you gather requirements from the software architect?' The results of this approach was remarkable. Where during our case study just 2 respondents indicate to collaborate on requirements gathering, this time 2/3rds did. A strong indicator that requirements gathering is an important collaborative activity in a wide variety of SPOs. Although both requirements prioritization and product roadmapping are collaborative activities for only half of all respondents, we believe this number exceeds the threshold to be called common. Despite its absence in previous literature, requirements refinement is a widespread collaborative activity within SPOs. This result generates opportunities for future research to examine the exact interaction between SPM, SA and the development team.

This time around, the UTAUT results are even less encouraging. Despite the changes made based on feedback for V1, respondents still indicate the prototype brings too little advantages to justify changing their approach. Our explanation for the questions concerning accomplishing tasks more quickly and increasing productivity mirrors the reason applicable for V1: utilizing AAMA means expanding your processes and inevitably slows you down. The lack of improvement as to whether AAMA is useful for their jobs is disconcerting. Our best efforts to reduce the mandatory overhead and create extra value had no effect.

There is, however, some inconsistency in respondents' answers. If AAMA is not useful for respondents' work and would not intend to use it within a year, then why do the majority of respondents think that AAMA improves information exchange between product managers and software architects and the specific analysis features are valuable? This discrepancy is an indication that respondents view AAMA as a 'nice to have' instead of a 'must have'. Although the parts do provide some form of value, the sum of all parts still offers insufficient value to consider AAMA a must have.

The answers to questions concerning effort expectancy were considerably less positive in comparison to the initial version. Surprising, considering that we extended significantly more attention to detail and smoothly guiding respondents through the prototype. Still, in comparison to the quality walkthroughs that contemporary SaaS business provide, our prototype might be considered rough around the edges. This argument notwithstanding, these considerably less positive results demonstrate people gain a better understanding of the systems when offered more extensive, oral explanations.

Chapter 8

Conclusions

Information exchange between product managers and software architects relies on primitive communicative methods: long meetings supported by impromptu drawings. A counterintuitive premise when you realize software architecture has large, direct impact on product success factors: creating a winning product and delivering value to customers. This thesis analyzes the interaction between product managers and software architects in an effort to identify ways to support their information exchange. To this end, we split this study into two research phases. (1) An exploratory case study to identify the critical processes with reciprocal information exchange. (2) Design science research based on the results from phase 1. By means of five semi-formal case studies we evaluate the initial design. Feedback from the case studies is used for a second iteration of the design science research. A survey among practitioners generates feedback and data to validate our earlier findings. In the next sections we present the primary results from these three phases, discuss the impact of our contributions and formulate future research opportunities.

8.1 Primary Results

At the start of this thesis, we elaborately discuss literature on SPM, SA and their interdependence at software producing organizations. We argue that product managers and software architects contribute to each other's goals in four activities: (1) requirements gathering, (2) prioritization, (3) refinement and (4) transferring product context.

Subsequently, we conduct five case studies with three goals: (1) to validate whether an interdependence between SPM and SA truly exists, (2) to identify which activities SPM and SA collaborate on and (3) explore if SPOs utilize any instrumentations to support information exchange. By elaborate analysis of the interviews, we confirm the existence of an interdependence between SPM and SA. Of the four activities identified in the literature study, requirements *gathering* and *refinement* are universally applicable to our case study SPOs. Product managers require software architects' technical requirement input to complement their functional lists. While software architects require requirement details from product managers to be able to properly develop new features. Prioritization is common but the information exchange is trivial and one-sided. Surprisingly, none of the case companies collaborate in (4) transferring product context. In subsequent research among 25 practitioners, we confirm that refinement and gathering are the primary collaborative activities for SPM and SA at SPOs. Just one case company utilized an instrumentation to facilitate information exchange between relevant stakeholders. All others rely on simple communicative methods: meetings supported by impromptu drawings of architectural models.

The second phase of this thesis consists of design science research, itself comprising three stages. Based on the results of the case study, we create the Accurate Architectural Models Approach (AAMA) that engages both the product manager and software architect to ensure ongoing accuracy and timeliness. The approach, however, requires further improvements to achieve continuous model accuracy. With this goal in mind, we envision an automatic tool that operationalizes AAMA and create a simple wire-frame prototype. Aside from continuous model accuracy, the prototype promises to prevent human error and enforce the architecture to become a living document.

To evaluate the prototype, we conduct 9 case studies with a product manager and software architect from 5 SPOs. From the interviews, we conclude that practitioners think product managers should not create architectural models, that the level of architectural detail is too high and that, despite these drawbacks, there is considerable value in being able to view connections between requirements and code components. Additionally, we notice that respondents are inclined to negatively respond to the strict approach guidelines.

Based on this feedback, we iterate and create a V2 of the prototype. V2 gives users considerably more freedom to choose when and how to create architectural models and what detail level to apply per requirement. On top of that, we change the presentation of the prototype to enhance focus on the feature-set instead of the accompanying approach. And not without success, this time only one respondent mentions that the prototype does not fit his organization's approach. Unfortunately, for all respondents but one, the feature set does not offer sufficient value to justify the necessary extra overhead.

8.2 Discussion

This thesis set out to answer the following main research question: *"How to support information exchange between Software Product Managers and Software Architects?"*. Based on our case studies and survey results, we conclude that the most important information exchange between SPM and SA takes place during requirements *gathering* and *refinement*. Despite our best efforts towards this goal, how to support the information exchange during these processes regrettably remains unclear.

This thesis introduced two versions of a tool to support the Accurate Architecture Model Approach (AAMA). The first version was a successful exploration for our ambitious goal of continuous architectural model accuracy. While respondents indicate the prototype is clear and understandable, the majority does not believe AAMA is useful for their work. Furthermore, they raise a number of legitimate concerns: product managers do not want to create architectural models and we require users to go into too much detail for AAMA to be practical. We address these issues in V2 by not dictating how to approach AAMA, instead allowing respondents to assess AAMA's value autonomously. In this regard, our efforts paid off. While nearly every interviewee for V1 mentioned that AAMA doesn't align with their current approach, just one mentioned it in his evaluation of V2. In theory, removing the distraction of misaligned approaches ensures that respondents focus on the prototype's features.

These improvements and new features, however, do not offer sufficient value for all respondents but one. When asked about expected performance and effort required, the majority maintains negative considerations. The inconsistency between these results and respondents' perceived value for product analysis and information exchange improvement prompts us to believe that respondents view AAMA as a 'nice to have' instead of a 'must have'. Although the features themselves might offer merit, to them the total benefits do not outweigh the required extra overhead. Based on these results, we provide a conclusive answer to our last research question: *"Do product managers and software architects intend to use our tool prototype?"* **No**.

At least, that is what respondents think based on a primitive prototype. The very nature of wireframes only permits users to experience the *"look and feel"*. To reliably evaluate practitioner's perception of AAMA, we need to conduct multiple, longitudinal case studies where we integrate a real, working prototype in an SPO's daily process. Only then are we able to evaluate the true potential of advanced features such as automatically identify critical components that need refactoring or requirements that can be easily implemented for relatively little cost. The possibilities for intelligent, automatic requirements insights are numerous. However, respondents' unwillingness to alter their processes in exchange for AAMA's theoretical benefits does not encourage us to continue down this road. If AAMA's benefits aren't significant enough currently, the odds remain small that a working prototype will cause respondents to view AAMA as a 'must have' instead of a 'nice to have'.

Instead, let's reconsider AAMA's premise: improve information exchange between product managers and software architects. The current solution attempts to incrementally improve communication between product managers and software architects by focusing on *software architecture*. Product managers, however, have such a strong focus on *functionality* that they are reluctant to embrace discussion models such as the FAM. This reluctance blinds product managers to the expected benefits: higher product quality due to improved feature development.

Furthermore, AAMA fails to solve any concrete problems these skeptical stakeholders have. A frequently uttered sentiment is that a respondent does not recognize or acknowledge they have any concrete information exchange problems: "we get along fine and finish our work eventually, right?" Despite the negative outcome of our efforts, thanks to this thesis we have gained valuable information on how to proceed. We now know that for AAMA to succeed, we need to define a concrete problem that AAMA solves for SPOs and clearly communicate why this problem requires solving in all SPOs. Only then, will we be able to create an approach and/or prototype that offers sufficient value to practitioners. Only then, will product managers and software architects acknowledge the necessity of change for their organization. To this end, our first future research objective is to concretely identify problems with requirements gathering and/or refinement. Depending on the results, we will proceed with the development of a working prototype and formulation of valuable analyses that capitalize on the connection between requirements and architecture.

8.3 Future Research

This thesis presents a first exploration of the interrelationship between software product management and software architecture in software producing organizations. The first author's future PhD will continue within this research domain to advance the industry's understanding of this particular interrelationship. Regrettably, not all phases of this research produced positive results. The garnered data and insights, however, provide us with the means to derive four, promising future research directions:

8.3.1 Formalize Requirements Gathering and Requirements Refinement

As identified in the preceding section, we need to determine concrete problems with the SPM and SA relationship to be able to create appropriate solutions. The first step to finding problems is fully understanding the intricate details of the SPM and SA relationship. Thanks to the results from both our case studies and survey we can confidently say that requirements gathering and refinement are central to the SPM and SA relationship. Therefore, to identify any concrete problems we will first diligently study and describe why and how product managers and software architects collaborate in requirements gathering and requirements refinement. A by-product of

this research is a better understanding of the requirement's life cycle within an SPO. Additionally, from this initial general understanding of how product managers work with architecture, we can construct a large scale survey to validate our results. Upon completion, we can formulate new approaches to solve these problems, if any in a similar manner to the design research presented in this thesis.

8.3.2 Why Do Software Producing Organizations Refuse to Utilize Architecture?

A recurring theme within the first case study was that interviewees did not create nor store any structured models to be used throughout the organization. Their reasoning being that it is *impossible* to maintain the timeliness and integrity of these models and that no models is better than inaccurate models. Yet, there are plenty of examples of organizations that document architecture to great success such as case company 4. This makes us wonder if there are other reasons why these organizations and their product managers refuse to utilize architecture models:

- Is the majority of their organization incapable of reading these models?
- Do underlying organizational issues undermine employees' discipline?
- Is the product manager's education and/or experience insufficient to work with architecture?
- Are product managers less interested in SA than we presume from logic?
- Do SPOs utilize other structural methods to connect requirements to code components or specific lines of code?
- Is the software architect already capable of AAMA's advanced analysis capabilities without additional software?

Answers to these questions provide us the means to further our understanding of how to study the SPM and SA interrelationship.

8.3.3 The SPM and SA Framework

Once we have collected sufficient knowledge on the intricacies of SPM and SA, we can begin to construct an SPM and SA framework. By recording how SPM and SA collaborate in each organization and whether they recognize any problems, we can generalize the results to identify organizational indicators for potential problems. Capturing these indicators in a framework then enables us to quickly diagnose new organizations and provide scientifically substantiated improvement advice.

8.3.4 Does Maturity Breed Success?

A major issue with the SPMCM by Bekkers et al. (2010a) is its lack of validity in a business sense. Despite that practitioners like it's guiding nature, the O&I department has long wondered how to validate its positive effects. Even how to determine whether the SPMCM has positive effects to begin with. Instead of attempting to measure and compare hard to come by numbers such as market share, revenue and profitability, we propose to measure a number of indirect indicators. The SPM and SA Framework will produce the first indicator: do mature SPOs according to the SPMCM also have a more mature SPM and SA relationship? In the coming years, expanding the variety and nature of these types of indirect indicators will draw an increasingly clear picture of the business value of SPMCM.

8.4 Acknowledgments

Writing has many rules, especially in academia. One of these concerns using the pronoun *I*. Strictly forbidden. Unless you crave global readership and succumb to the dark arts: popular science. But for some reason this rule also does not apply to the acknowledgments section of a thesis. All of a sudden, the academic collective acknowledges the *I* in academ*i*a.

Unfortunately, "Oh, yes, the *acknowledgements*. I think not. I did it. I did it all, by myself" (Olin Shivers, 1994) does not apply. The initial seed of this thesis was planted by Sjaak Brinkkemper. Afterwards, he continuously cultivated the intricate details of our seedling to ensure a specific level of quality. However, this thesis would have been abysmal if not for the endless support and patience of Jan Martijn van der Werf. He must have read every single word on these pages at least five times. His ability to produce new questions that demand answers is uncanny. Kevin Vlaanderen his diligent reviews inevitably lead to firm, constructive feedback that elevates the quality of one's writing. Jaap Kabbedijk is fun to throw balls at.

Software is eating the world. Make sure it doesn't break.

References

Avgeriou, P., Grundy, J., Hall, J. G., Lago, P., & Mistrk, I. (2011). *Relating software requirements and architectures* (1st ed.). Springer.

Bekkers, W., Spruit, M., van de Weerd, I., van Vliet, R., & Mahieu, A. (2010b). A situational assessment method for software product management. In *Proceedings of ecis* (pp. 22–34). Pretoria, South-Africa.

Bekkers, W., van de Weerd, I., Spruit, M., & Brinkkemper, S. (2010a). A Framework for Process Improvement in Software Product Management. In *Proceedings of eurospi* (pp. 1–12).

Brinkkemper, S., & Pachidi, S. (2010). Functional architecture modeling for the software product industry. In *Software architecture* (Vol. 6285, p. 198-213). Springer.

Chen, C., Shao, D., & Perry, D. (2007, May). An exploratory case study using cbsp and archium. In *Second workshop on sharing and reusing architectural knowledge - architecture, rationale and design intent* (p. 3-3). IEEE.

Chitchyan, R., Pinto, M., Rashid, A., & Fuentes, L. (2007). Compass: Composition-centric mapping of aspectual requirements to architecture. In *Transactions on aspect-oriented software development iv* (Vol. 4640, p. 3-53). Springer.

Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., & Little, R. (2002). *Documenting software architectures: Views and beyond.* Pearson Education.

Condon, D. (2002). *Software product management.* Boston, Massachusetts: Aspatore Books.

Corbetta, P. (2003). *Social Research: Theory, Methods and Techniques.* London: Sage Publications.

Davis, A. M. (2005). *Just enough requirements management: Where software development meets marketing.* New York, NY, USA: Dorset House Publishing Co., Inc.

Driessen, V. (2010, January). *A succesful git branching model.* Retrieved from `http://nvie.com/posts/a-successful-git-branching-model/`

Ebert, C. (2007). The Impacts of Software Product Management. *Journal of Systems and Software*, *6*(80), 850–861.

Ebert, C. (2009). Software Product Management. *Crosstalk*, *22*(1), 15–19.

Ebert, C., & Brinkkemper, S. (2014). Software product management an industry evaluation. *Journal of Systems and Software*(0), -. doi: 10.1016/j.jss.2013.12.042

Fontana, F. A., & Zanoni, M. (2011). A tool for design pattern detection and software architecture reconstruction. *Information Sciences*, *181*(7), 1306 - 1324. doi: 10.1016/j.ins.2010.12.002

Fricker, S. (2012). Software product management. In *Software for people* (p. 53-81). Springer.

Gorchels, L. (2000). *The Product Manager's Handbook: The Complete Product Management Resource (2nd edition)* (2nd ed.). Columbus, OH, USA: McGraw-Hill.

Grünbacher, P., Egyed, A., & Medvidovic, N. (2004). Reconciling software requirements and architectures with intermediate models. *Software and Systems Modeling*, *3*(3), 235-253. doi: 10.1007/s10270-003-0038-6

Hall, J. G., Jackson, M., Laney, R. C., Nuseibeh, B., & Rapanotti, L. (2002). Relating software requirements and architectures using problem frames. In *Ieee international requirements*

engineering conference (re'02) (p. 137-144). doi: 10.1109/ICRE.2002.1048516

Hartl, M. (2014, June). *Ruby on rails tutorial.* Retrieved from `http://www.railstutorial.org/book`

Helferich, A., Schmid, K., & Herzwurm, G. (2006, December). Product management for software product lines: An unsolved problem? *Communications of the {ACM}*, *49*(12), 66–67. doi: 10.1145/1183236.1183268

Hevner, A. R., & Chatterjee, S. (2010). Design Science Research Frameworks. In *Design research in information systems* (pp. 23–31). New York, NY, USA: Springer.

ISO, J. (2011a). IEC 25010: 2011: Systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models. *International Organization for Standardization*.

ISO, J. (2011b). Iec 42010:2011, systems and software engineering architecture description. *International Organization for Standardization*.

Jansen, A., & Bosch, J. (2005). Software architecture as a set of architectural design decisions. In *Proceedings of wicsa '05* (p. 109-120). doi: 10.1109/WICSA.2005.61

Jansen, S., & Brinkkemper, S. (2008). Applied Multi-Case Research in a Mixed-Method Research Project: Customer Configuration Updating Improvement. In A. Cater-Steel & L. Al-Hakimi (Eds.), *Information systems research methods, epistemology, and applications* (pp. 120–139). Utrecht: IGI Global.

Kajornboon, A. (2005). Using Interviews as Research Instruments. *E-Journal for Research Teachers*, *2*(1).

Karlsson, L., sa G. Dahlstedt, Regnell, B., och Dag, J. N., & Persson, A. (2007). Requirements engineering challenges in market-driven software development an interview study with practitioners. *Information and Software Technology*, *49*(6), 588 - 604. doi: http://dx.doi.org/10.1016/j.infsof.2007.02.008

Khurum, M., & Gorschek, T. (2011). A method for alignment evaluation of product strategies among stakeholders (mass) in software intensive product development. *Journal of Software Maintenance and Evolution: Research and Practice*, *23*(7), 494–516. doi: 10.1002/smr.536

Kittlaus, H.-B., & Clough, P. N. (2009). *Software Product Management and Pricing: Key Success Factors for Software Organizations.* Berlin, Germany: Springer.

Kruchten, P. (1995). The 4+1 view model of architecture. *IEEE Software*, *12*(6), 42-50. doi: http://doi.ieeecomputersociety.org/10.1109/52.469759

Maier, M. W., Emery, D., & Hilliard, R. (2001, Apr). Software architecture: Introducing ieee standard 1471. *Computer*, *34*(4), 107-109. doi: http://doi.ieeecomputersociety.org/10.1109/2.917550

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*(4), 251 - 266. doi: http://dx.doi.org/10.1016/0167-9236(94)00041-2

Mirakhorli, M., & Cleland-Huang, J. (2013). Traversing the twin peaks. *IEEE Software*, *30*(2), 30-36. doi: http://doi.ieeecomputersociety.org/10.1109/MS.2013.40

Nuseibeh, B. (2001, March). Weaving together requirements and architectures. *Computer*, *34*(3), 115–119. doi: 10.1109/2.910904

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: A roadmap. In *The future of software engineering* (pp. 35–46). New York, NY, USA: ACM. doi: 10.1145/336512.336523

Potts, C. (1995). Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software. In *Proceedings of the second ieee international symposium on requirements engineering.*

Regnell, B., Berntsson Svensson, R., & Olsson, T. (2008). Supporting roadmapping of quality requirements. *Software, IEEE*, *25*(2), 42–47.

Rozanski, N., & Woods, E. (2011). *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Boston: Addison-Wesley.

Salfischberger, T., van de Weerd, I., & Brinkkemper, S. (2011, Aug). The functional architecture framework for organizing high volume requirements management. In *Software product management (iwspm), 2011 fifth international workshop on* (p. 17-25). doi: 10.1109/IWSPM.2011.6046208

Sawyer, P., Sommerville, I., & Kotonya, G. (1999). Improving market-driven re processes. In *Proceedings of the international conference on product focused software process improvement* (Vol. 195, pp. 222–236). Oulu, Finland.

Schmidt, F., MacDonell, S., & Connor, A. (2012). An automatic architecture reconstruction and refactoring framework. In R. Lee (Ed.), *Software engineering research, management and applications 2011* (Vol. 377, p. 95-111). Springer.

Taylor, R. N., Medvidovic, N., & Dashofy, E. M. (2010). *Software architecture: Foundations, theory, and practice*. John Wiley & Sons.

van der Aalst, W. (2009). Using process mining to generate accurate and interactive business process maps. In W. Abramowicz & D. Flejter (Eds.), *Business information systems workshops* (Vol. 37, p. 1-14). Springer.

van der Hoek, A., Hall, R. S., Heimbigner, D., & Wolf, A. L. (1997, November). Software release management. *SIGSOFT Softw. Eng. Notes*, *22*(6), 159–175. doi: 10.1145/267896.267909

van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L. (2006). On the creation of a reference framework for software product management: Validation and tool support. In *International workshop on software product management* (p. 3-12). doi: 10.1109/IWSPM.2006.6

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly*, *27*(3), pp. 425-478.

Vlaanderen, K., Jansen, S., Brinkkemper, S., & Jaspers, E. (2011). The agile requirements refinery: Applying {SCRUM} principles to software product management. *Information and Software Technology*, *53*(1), 58 - 70. doi: http://dx.doi.org/10.1016/j.infsof.2010.08.004

Vogl, H., Lehner, K., Grunbacher, P., & Egyed, A. (2011, Aug). Reconciling requirements and architectures with the cbsp approach in an iphone app project. In *Requirements engineering conference (re), 2011 19th ieee international* (p. 273-278). doi: 10.1109/RE.2011.6051625

Ward, P. T., & Mellor, S. J. (1986). *Structured development for real-time systems: Vol. i: Introduction and tools*. Prentice Hall.

Whalen, M. W., Gacek, A., Cofer, D., Murugesan, A., Heimdahl, M. P. E., & Rayadurgam, S. (2013). Your "what" is my "how": Iteration and hierarchy in system design. *IEEE Software*, *30*(2), 54-60. doi: http://doi.ieeecomputersociety.org/10.1109/MS.2012.173

Wnuk, K., Regnell, B., & Schrewelius, C. (2009). Architecting and coordinating thousands of requirements an industrial case study. In *Requirements engineering: Foundation for software quality* (Vol. 5512, p. 118-123). Springer.

Xu, L., & Brinkkemper, S. (2007). Concepts of Product Software. In *European journal of information systems* (Vol. 16, pp. 531–541).

Yin, R. K. (2009). *Case Study Research - Design and Methods*. SAGE Publications.

Appendices

Appendix A

Case Study Protocol I

Introduction consisting of:

- overview of the research goals
- defining key concepts of Software Product Management (SPM) and Software Architecture (SA)
- explaining that the interview is divided over three sections: role, processes and tools.

A.1 Case Study Questions

1. What is the role of SPM in the organization?
2. What is the role of SA in the organization?
3. How do SPM and SA collaborate in the organization?
4. What is the or are the essential aspects of their collaboration?
5. What are problems and/or impediments to their collaboration?
6. Does the organization utilize any instrumentations to facilitate their collaboration?
   (a) What problems do these instrumentations solve?
   (b) Do the interviewees consider these instrumentations efficient?
   (c) Do the interviewees consider these instrumentations effective?
7. What other tools do the interviewees consider to be appropriate for their organization?

A.2 Outline Case Study Report

1. Discuss SA and SPM processes in the organization
2. Summarize the most important processes:
   (a) process overview
   (b) process purpose
   (c) process impediments
3. Summarize the tools used in the organization:
   (a) tool overview
   (b) tool purpose
   (c) tool impediments

A.3 Data Collection Procedures

Our case studies consist of interviews with both a (lead) product manager and a software architect. We expect to collect data on their collaborative efforts and instrumentations they currently utilize to facilitate their information exchange. If possible, we also review internal process specifications for their collaboration.

- Respondent info 1
- Respondent info 2
- Respondent info 3
- Respondent info 4
- Respondent info 5

A.4 Example Case Study Report - COMPANY NAME

COMPANY NAME creates software to automate system administration. They have three products: workspace management, automation manager and IT Store. Together they allow an IT department to automatically manage desktops, create tasks to automatically install new software or upgrades and empower users to easily install their new, manager-permitted applications. The interviewer talked to the lead SPM (SPM NAME) and their most senior SA (SA NAME).

A.4.1 SPM & SA Processes

The SPM's role is to figure out and decide what the company is going to do with their products. What are we going to add and what are we going to remove. Everything has to do with that, directly or indirectly. The role is primarily focused on marketing, deciding 'what' should be done, letting R&D figure out the 'how'. The SA is primarily responsible for $PRODUCTNAME$, which is a product he helped establish. He feels like the technical product owner of $PRODUCTNAME$, similar to how you have functional product owners in SPM. Aside from thinking about architecture, he does some engineering and wants to be involved when customer issues arise, especially those that escalate. There are two types of SA and SPM collaboration: discussing desirables and feasibles. These are spread out over several meetings in which current plans are discussed.

Desirable - There is no official forum that decides which requirements to add to the backlog, this is the sole responsibility of the product manager and he typically is the primary contributer of requirements that end up on it. However, the day to day collaboration of PM and SA is frequent enough that decisions are discussed. Furthermore, the SA is an important internal stakeholder whose input is actively solicited. All resulting user stories are registered according to a template in OnTime.

Feasibility - After three types of meetings, desired requirements are refined into developer-ready user stories. Once a month, the PM team presents its initial ideas to technical stakeholders. In 50% of the cases, these initial ideas need more contents to be clear to them. When a requirement is clear, it is discussed during a biweekly backlog meeting between the PM and SA. Here, requirements are discussed on a high level - 'this is easy', 'this is impossible', 'do you want to go left or right on this feature?'. During this process, the feasibility of each requirement is assessed and will sometimes result in its exclusion. Finally, every week all stakeholders refine requirements to enable the developer to estimate the time required to build a feature. The developer that is going to build the feature is present so he can make a subjective estimate. When a requirement has been discussed at all three meetings, the amount of details in OnTime should be sufficient for the developer to build it autonomously.

A.4.2 Important Processes

Process 1: Desirables
The PM team assembles a list of features that he would like to include in the next release. They collect input from internal and external stakeholders. Feature feasibility is discussed with tech leads on an ad-hoc basis such as when a feature is particularly technical.
Purpose: Identifying, prioritizing and selecting requirements to add to the backlog.
Impediments: The PM team is still learning, leading to underspecified requirements.

Process 2: Requirements meeting
Once a month PM presents new features to all technical stakeholders.
Purpose: gathering first feedback on PM ideas, validating whether idea is clear enough.
Impediments: n/a

Process 3: Backlog meeting
Every other week the PM and SA review items on the backlog to discuss them on a high level.
Purpose: documenting whether features are feasible or not.
Impediments: The PM has limited knowledge of how the products are implemented on a technical level. As a consequence, the PM is bound to have ideas which are not possible. To explain why, the SA will draw on a whiteboard.

Process 4: Requirements refinement
Every week the PM, SA, Requirements Engineering and development team look at requirements in detail.
Purpose: refine coarse-grained requirements that have been selected for a sprint into fine-grained specified requirements so that the developer who is going to build the feature knows exactly what is necessary and can estimate the time necessary to build it.
Impediments: n/a

A.4.3 Tools

Tool 1: OnTime
OnTime is an Agile Project Management Tool which the organization primarily uses to track requirements.
Purpose: For PM, requirements 'live' in OnTime. PM enters them into the system after which each requirement is enriched with more details, mock-ups and/or emails. RE registers developer questions in OnTime. When a requirement is sufficiently enriched, it is exported to a feature tracker used by the developers.

Tool 2: Team Foundation Server (TFS)
For this organization, TFS is a source code management system and project management tool for the developers.
Purpose: Overview of requirements for R&D, subdivided in sprints.

Tool 3: User Story Template
New requirements are specified according to a standardized user story template.
Purpose: Uniform requirement registration.

Appendix B

Case Study Protocol II

The case studies and surveys follow the following steps

- Introducing research goals
- Explaining research structure
- Gather contextual information
- Guide the respondent through the prototype
- Allow the respondent to evaluate the prototype freely
- Present the survey to the respondent

B.1 Case Study Questions

1. Contextual questions
   (a) How many years of experience does the respondent have as product manager/software architect?
   (b) How does the respondent estimate the product manager's technical competence in comparison to the software architect's?
   (c) What are the collaborative activities between the product manager and software architect?
2. Closed Questions
   (a) Does respondent think AAMA is useful for his job?
   (b) Does respondent think AAMA enables him to accomplish his tasks more quickly?
   (c) Does respondent think AAMA increases his productivity?
   (d) Does respondent think AAMA increases his chances of getting a raise?
   (e) Does respondent think his interaction with AAMA would be clear and understandable?
   (f) Does respondent think it would be easy for him to become skillful at using AAMA?
   (g) Does respondent think he would find AAMA easy to use?
   (h) When does respondent intend to use AAMA (presuming that a ready version is available)?
3. Open Questions
   (a) What are the respondent's initial thoughts on the prototype?
   (b) Does the respondent think AAMA will improve his information exchange with software architect/product manager?
   (c) Does the respondent bring up any other applications for AAMA which are valuable to him?

B.2 Outline Research Report

1. Contextual information
2. Performance Expectancy
3. Effort Expectancy
4. Behavioral Intention
5. AAMA evaluation

B.3 Data Collection Procedures

The case studies consist of separate interviews with a product manager and software architect from each company. We expect to collect data on their collaborative activities, their thoughts when they are clicking through the prototype and an elaborate evaluation of the AAMA prototype upon completion.

- Respondent info 1
- Respondent info 2
- Respondent info 3
- Respondent info 4
- Respondent info 5

Appendix C

Survey Questions

1. Contextual questions
   (a) Years of experience as a software product manager/architect
   (b) What continent are you from?
   (c) The product manager's technical competence in comparison to a software architect is ... on a scale from 1 to 7
   (d) Do you gather requirements from the architect?
   (e) Do you have formalized meetings with (amongst others) the software architect to refine requirements?
   (f) Do you create product roadmaps together with the software architect?
   (g) Do you prioritize requirements into sprints together with the software architect?
   (h) Can you think of any other collaborative activities?
   (i) Performance Expectancy
2. Closed questions
   (a) I think AAMA is useful for my job
   (b) Using AAMA enables me to accomplish my tasks more quickly
   (c) Using AAMA increases my productivity
   (d) My interaction with AAMA would be clear and understandable
   (e) It would be easy for me to become skillful at using AAMA
   (f) I would find AAMA easy to use
   (g) I intend to use AAMA in the next n months
3. Open Questions
   (a) What are your initial thoughts on the prototype?
   (b) Do you think AAMA will solve a problem for you?
   (c) Do you think AAMA will improve the information exchange between you and product managers/software architects?
   (d) Is requirements clustering as seen on the last page valuable to you?
   (e) Is critical components as seen on the last page valuable to you?
   (f) What other requirement insights that can be obtained by connecting requirements and code components can you think of that are of value to you?

Appendix D

8th International Workshop on Software Product Management Paper

Alignment of Software Product Management and Software Architecture with Discussion Models

Garm Lucassen, Jan Martijn E. M. van der Werf and Sjaak Brinkkemper
Department of Information and Computing Sciences
Utrecht University
Princetonplein 5, 3584 CC Utrecht, The Netherlands
{g.lucassen, j.m.e.m.vanderwerf, s.brinkkemper}@uu.nl

***Abstract*—How to achieve alignment of software product management with software architecture and whether there is a business case for doing so is scientifically unknown. Yet, software architecture has large, direct impact on product success factors: creating a winning product and delivering value to customers. In this exploratory case study paper we identify the most critical processes for SPM and SA alignment: requirements gathering and refinement. These processes require effective communication supported by high level architectural views. Our respondents, however, rely on simple methods due to their negative experiences with formalized models. Based on these findings, we propose the Accurate Architectural Models Approach (AAMA) which prevents architectural model divergence.**

I. Introduction

Software product managers achieve product success by focusing on three goals: (1) creating a winning product and business case, (2) conquering markets and growing market share and (3) delivering value to customers [1]. An essential aspect to reaching these goals is requirements engineering (RE) [1]. Software architects have similar goals. They make architectural design decisions (ADDs) based on available requirements and previous ADDs [2] to satisfy stakeholders' functionality and quality attribute requirements [3]. Believing that an early software architecture (SA) understanding is as important to project success as requirements understanding, [4] adapted the *Twin Peaks Model* as introduced by Ward [5]. The Twin Peaks model supports the co-evolution of SA and RE by concurrently developing requirement details and architectural specification (Figure 1).

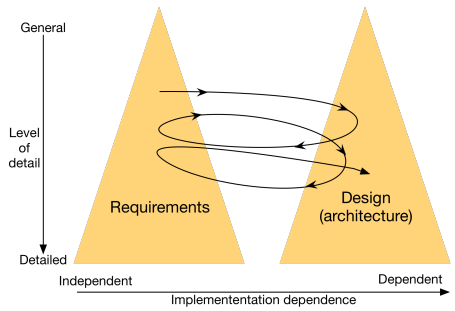


Fig. 1. The Twin Peaks of Requirements and Architecture [4]

The Twin Peaks model originates from research on software *projects*. However, the *product* software domain diverges from projects. As software producing organizations (SPOs) grow, they encounter large numbers and varieties of stakeholders that generate a continuous input of requirements [6]. Consequentially, decisions need to be made on which requirements to fulfill at what moment in time. To manage these inputs and decisions, SPOs require special processes [7], [8] that are typically the responsibility of *software product managers*. According to [9], product managers are crucial to product and company success and [1] even states that "companies win or fail depending on their product managers".

For SPOs, product managers are the primary interface and representative of stakeholders' requirements to the development team including the architect. Consequentially, we hypothesize that successful collaboration between software product management (SPM) and software architecture (SA) is essential for achieving both product success goal 1 and 3 and therefore product success. [10] supports this premise, noting that SPM alignment with SA is critical to software product line success. However, no publications to substantiate this claim are available. Before we can attempt to confirm the importance of SPM and SA interdependence, we first need to establish whether SPM and SA at contemporary SPOs collaborate in the first place. To this end, we pose the following research question: *"How do software product managers and software architects collaborate?"*.

This exploratory research discusses five case studies at Dutch SPOs to detail their SPM and SA processes and facilitating instrumentations. We identify two critical processes where product managers and architects exchange information: requirements gathering and requirements refinement. These activities require reciprocal information exchange on a high technical level to drive product success. Respondents, however, lack a structured approach that warrants SPM and SA alignment. We propose a solution that prevents architectural model divergence to improve SPM and SA alignment.

The subsequent sections II, III and IV discuss relevant SA and SPM literature. Next, we detail the research approach of this paper in section V. Section VI presents the case study results, followed by a discussion in VII and a proposed solution in VIII. We summarize our primary results and present future research opportunities in the concluding Section IX.

II. Software Architect Processes

Software Architecture is "the set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both" [3]. Typical software architecture documentation consists of a collection of views, which are "a representation of a set of system elements and the relationships associated with them" [3]. An organization creates architecture documentation for multiple purposes. Organizations use it to design, analyze, communicate and/or educate a software system from multiple, varying stakeholder role perspectives. Considering that stakeholders can be as diverse as a customer and a developer, the seminal standard on Software Architecture (IEEE Standard 1471-2000) declares that architectural models are inherently multi-viewed [11].

A software architect has four primary tasks: developing project strategy, designing systems, communicating with stakeholders and being a leader [12]. Over time, the architect develops a software architecture by making architectural design decisions (ADDs) based on the project context which consists of stakeholder requirements and previous ADDs [2]. The ADDs in aggregate attempt to satisfy stakeholders' quality attribute requirements [3]. Quality attribute requirements specify system operation, contrary to functional requirements which specify system behavior. Their importance for product success is equal. After all, a system's ability to produce correct information is useless when the system is unable to deliver that information in time, securely or at all [3]. Achieving quality attribute criteria is central to the software architecture discipline. The ISO/IEC 25010 standard on system and software quality models posits that software product quality attributes comprise eight characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability [13]. Each characteristic contains a number of related sub-characteristics, which "are useful for specifying requirements, establishing measures, and performing quality evaluations" [13].

The ADDs a software architect makes are based on stakeholder requirements. Therefore, quality attributes are an indirect consequence of the project's set of requirements, creating a strong relationship between RE and SA. Drawing inspiration from Nuseibeh's Twin Peaks model for software development [4], several authors have formulated approaches to further relate software requirements and architecture [14]–[18]. However, none of these are widely adopted in business or academia. For example, the CBSP (Component-Bus-System-Property) approach by Grunbacher [15], which delivers a "proto-architecture" based on functional requirements to prescribe further architectural development, has been applied a limited number of times since its introduction [19], [20].

III. Software Product Processes

Software product management (SPM) is "the discipline governing a software product over its whole life cycle, from its inception to customer delivery, in order to generate the biggest possible value to the business" [22]. [23] defines a software product as "a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market". As software producing organizations (SPOs) grow, they encounter large numbers and varieties of stakeholders that generate a continuous input of requirements [6]. Due to limited resources, decisions need to be made on which requirements to fulfill at what moment in time.

To manage these inputs and decisions, SPOs require special processes [7], [8] that are typically the responsibility of *software product managers*. A product manager attempts to achieve product success by focusing on three goals: (1) creating a winning product and business case, (2) conquering markets and growing market share and (3) delivering value to customers [1]. In a case study examining what product managers should do to reach these goals, practitioners assert that "he does everything the product needs to be successful" [24]. Nevertheless, theory stipulates they manage requirements, produce release definitions and define products in an environment with many internal and external stakeholders [25], [26]. This wide range of responsibilities makes product managers crucial to product and company success [9], [27].

Although the number of publications on SPM is growing each year, the field is still young [28]. In 2011, [29] could find only 25 articles on the subject. Consequently, SPM scholars have difficulty to provide scientifically validated best practices to industry. However, several authors have developed SPM reference frameworks to exactly this end [21], [22], [25], [30]. Of these, the Software Product Management Competence Model (SPMCM) by [25] and extended by [21] is based on extensive empirical research. The SPMCM (Figure 2) presents the four main business functions within SPM: requirements management, release planning, product planning and portfolio management. Each business function consists of a number of focus areas that represent a strongly coherent group of capabilities. In theory, SPOs that hold these capabilities are more effective in their product management but this remains unproven to date.

[31] presents an approach to *agile* SPM, introducing product management sprints alternating with development sprints to ensure requirements' are ready for use when software development starts. During a product management sprint, the SPM team conducts *requirement refinement* for product backlog requirements from coarse-grained concepts to fine-grained instructions that software engineers use. Concretely, the product manager refines vision into specified requirements through three, progressively more fine-grained stages: themes, concepts and requirement definitions.

A. Requirements Management

Of the four main business functions in the SPM Competence Model, *requirements management* (RM) is the most frequent, lowest level function a product manager partakes in. Coincidentally, it is the most extensively studied domain in IS/IT research. [21] identifies three focus areas relevant in the product software context: (1) requirements gathering: "the acquisition of requirements from both internal and exter-

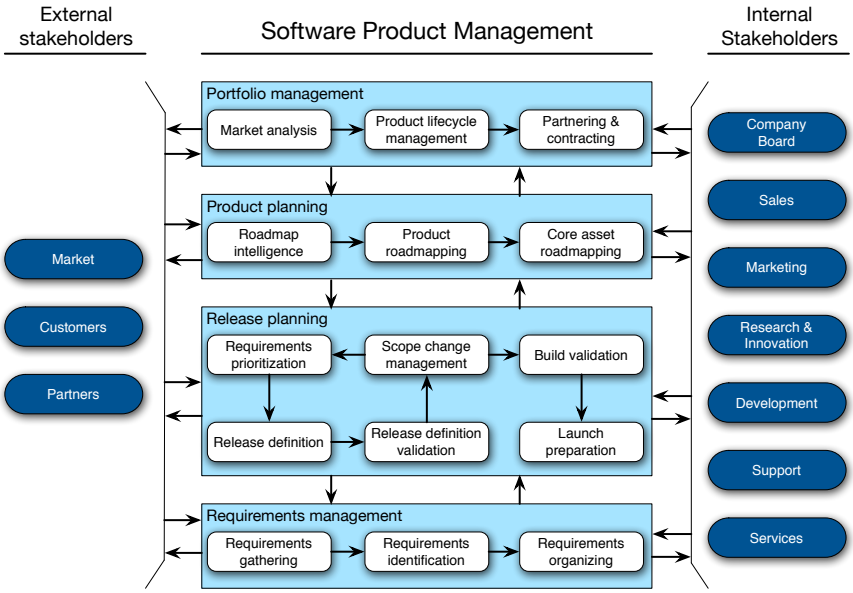


Fig. 2. Software Product Management Competence Model [21]

nal stakeholders", (2) requirements identification: "identifying the actual product requirements by rewriting the market requirements to understandable product requirements, and connecting requirements that describe similar functionality" and (3) requirements organizing: "structuring the requirements throughout their entire lifecycle based on shared aspects, and describing the dependencies between product requirements". These three activities combined with prioritizing requirements are traditionally called *requirements engineering* [32] and considered to be a core product manager responsibility [1], [26].

Requirements engineering typically occurs early in the lifetime of a project to prevent increased costs associated with requirements errors later on [33]. The product software context, however, presents specific challenges that necessitate a different approach to RE [34]–[36]. The large number of requirements makes RE a daily necessity over the lifespan of a software product. Furthermore, the large number of requirements frequently produces complications such as *information overload, combinatorial explosions* and *over-scoping* [7]. Several approaches to prioritize large numbers of requirements are available, both generic [37], [38], and context specific [39]. None of these are widely adopted by SPOs, despite providing concrete advantages over traditional prioritization techniques.

B. Release Planning, Product Planning, Portfolio Management

The emphasis of this research lies on requirements management, because it is the most frequent activity a product manager conducts. Therefore, this subsection details the remaining SPM activities only briefly. (1) *Release planning* comprises activities for creating and launching a release [21]. The broad business function comprises a variety of tasks ranging from prioritizing and selecting requirements to preparing the launch with internal and external stakeholders through negotiations and trainings. (2) A product manager conducts *product planning* to construct a roadmap, which is an action plan for a specific time period. The business function consists of three activities: roadmap intelligence, product roadmapping and core asset roadmapping. Information gathered in the first activity is input during actual roadmap construction. (3) *Portfolio management* consists of gathering strategic information and making product decisions across the entire portfolio [21]. Decision support information gathered during market analysis contributes to the business's decision making on product lifecycle management and partnering & contracting.

IV. Software Product Manager and Software Architect Interdependence

Of architects' primary tasks, product managers are involved in two: *developing project (product) strategy* and *communicating with stakeholders*. While SPM has a strong functional focus, SA is primarily responsible for balancing the appropriate quality attributes (QAs). While architects are able to formulate adequate technical solutions for any functional requirement, their proposed solution might not be optimal considering product context [12]. For example, the QA *performance efficiency* might be imperative for generating customer reports, while *reliability* has priority for the support interface. It is the product manager's responsibility to accurately convey context to enable the architect to make the right architectural decisions. Depending on the product manager's background (30-50% marketing or engineering [1]) and position (R&D, marketing or P&L [27]), the architect can discuss QA specifics with the product manager. Besides technical stakeholders, [12] notes that architects are in frequent contact with customers and users. For SPOs, however, the large quantity of diverse customers and users poses a communication challenge. Instead, the product manager acts as a representative on both sides. He communicates customer requirements to the architect, receives feedback and reports results to the customer.

Of the three product success goals introduced in section III, the architect has direct impact on (1) creating a winning

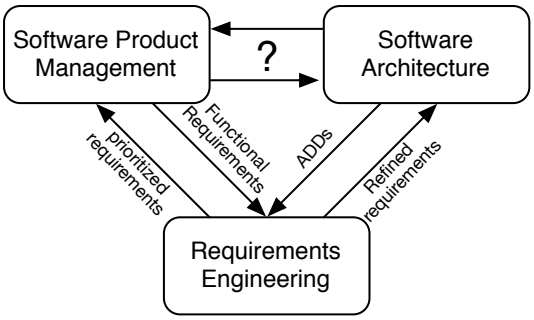


Fig. 3. SPM, RE and SA relationships

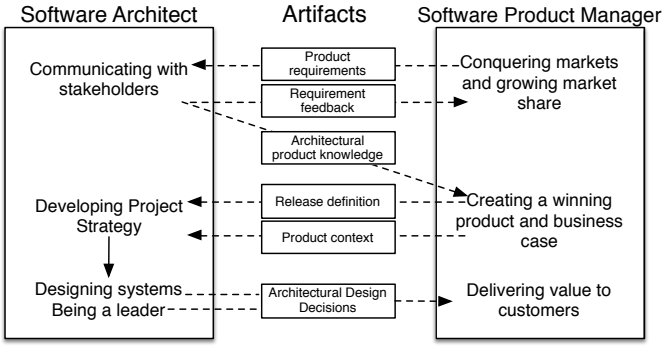


Fig. 4. Reciprocal Contributions to SPM and SA goals

product (and business case) and (3) delivering value to customers. Initially, we speculated that the product manager assumes the left peak of the Twin Peaks model. He exchanges functional details to refine requirements concurrently with the architect's specifications. In a previous case study however, the product manager's requirement refinement scope is limited: "The detailed definition of requirements is performed in three steps, of which only the first one is performed by the SPM team(s) [...] The software development teams then elaborate these [high-level requirement definitions] into requirements containing a detailed description of some desired functionality, described in sufficient detail to work with" [31]. The SPMCM by [21] does not recognize requirement refinement as an SPM activity to begin with. Instead, we believe there is a conceptual relationship between three activities: SPM, RE and SA. SPM is primarily responsible for high-level strategic issues such as market analysis, product roadmapping and fostering high-level requirements. RE refines the high-level requirements into sufficient detail to enable SA to realize the actual end-product. The information exchange between SA and SPM and their respective impact on one another's activities, however, is unknown. Figure 3 displays the conceptual relationship between SPM, RE and SA and illustrates this paper's central question: how do SPM and SA collaborate?

Requirements' central role for both stakeholders is a recurring theme in this paper. Within the SPMCM, product managers involve SA during requirements gathering, identification, organizing, and prioritization. Architects require SPM input to refine requirements, formulate correct architectural approaches and communicate with external stakeholders. Both the product manager and architect have a common goal: satisfy customer requirements to deliver value and create product success. Reaching this goal demands selecting the right requirements for a release and enabling developers to implement requirements in the right manner. While organizing, identifying requirements and formulating architectural approaches are essential, we believe these activities are out of the stakeholder's respective scope. Based on these literature findings, we posit that product managers' and architects' contribute to each other's goals in four activities: (1) requirements gathering, (2) prioritization, (3) refinement and (4) transferring product context. In terms of the conceptual relationship in Figure 3, the product manager provides product context and requirement details to the architect in exchange for architectural expertise in order to enhance one another's decision making.

We elaborate on the conceptual relationship by taking the perspective of SPM and SA their primary goals as introduced in sections II and III (Figure 4). The goals are connected with artifact flows we believe product managers and architects to exchange. Note that you should not read these flows as a linear route, but as an indefinite, iterative process. The result displays the mutual contributions to their respective goals, which we detail in this paragraph. As the primary interface for internal and external stakeholders towards the architect, the product manager receives a continuous stream of market requirements and customer requests. He organizes and selects specific *product requirements* to present to the architect. The architect responds with *requirement feedback*, which the product manager communicates to the appropriate stakeholders with the intention to conquer markets and grow market share. On the second level, the architect transfers *architectural product knowledge* to the product manager to enable collaborative requirements gathering, identification, organization, prioritization and refinement. The result is a mature *release definition*, which the architect combines with *product context* to *develop a project strategy*. Next, the architect leads the development team to design the system, making *architectural design decisions* based on the project strategy containing all provided inputs. These architectural design decisions lead to a finished product release and in turn deliver value to the customer.

To support reciprocal information exchange, product managers and architects need architecture views that relate requirements and architecture by displaying information and data on the intersection of their respective responsibilities. Unfortunately, the approaches relating requirements and architecture mentioned in Section II do not satisfy this need. Their goal is to facilitate software architecture design by developing functional requirements models. Identifying a similar disparity, [40] introduced the Functional Architecture Framework (FAF): an architectural view instrumentation that aims to support requirements management for SPOs with high requirement quantities. By linking requirements to specific

architectural application modules, requirement responsibility can immediately be assigned to the correct individual or team. To use the FAF, the SPM must first describe the product's functional architecture with a graphical architectural description language called the *Functional Architecture Model* [41]. It consists of a product context, a product scope that includes all high granularity application modules and typically two to three layers of sub-models. Next, the product manager assigns all existing requirements to the appropriate submodules and allocates newly incoming requirements in a similar manner. Finally, when a functional requirement enters development, the architect links his implementation solution enabling the product manager to analyze which functional requirements share a common technical component.

V. Case Study Research Approach

The previous sections explicated why we believe that collaboration between SPM and SA is crucial for product success. However, no empirical research that substantiates this claim by speaking to practitioners is available. As a first step, we answer the research question *"How do software product managers and software architects collaborate?"* by conducting five case studies with product managers and architects from five SPOs. Each SPO originates from the Netherlands, but their age, size and target market(s) differentiate. One SPO is an autonomous subsidiary of a bank with several small technical teams, while another is a 30 year old multinational with 2000 employees buildings software supporting SMEs. Each case study consists of an interview of approximately 1,5 hours with both a (senior) product manager and (senior) architect.

Due to the exploratory nature of this research, a formally structured interview or standardized questionnaire was unsuitable [42], [43]. Instead, we held semi-structured interviews to allow respondents to speak freely and the interviewer to pose follow-up questions, while maintaining the ability to compare responses. For each interview, the end goal was to establish how the product manager and architect at the case study company collaborate. To this end, the interviewer posed 7 questions such as: *"What is the role of SPM in the organization?"* and *"Does the organization utilize any instrumentations to facilitate their collaboration?"*.

After each interview, the interviewer put together a case study report, describing the product manager and architect's role in the organization, how both roles collaborate, whether either experienced any problems in this collaboration and what kind of instrumentations they utilize. Based on the case study report, the interviewer records whether the respondents collaborate in the four collaborative activities.

Before we commenced our case studies, we examined [44]'s guidelines for case study research. Our approach complies with the three conditions to determine when a case study is appropriate for your research: (1) we pose a 'how' research question, (2) we do not have control of behavioral events and (3) we focus on contemporary events. Furthermore, the research approach adheres to the criteria for research design quality as follows: we use multiple sources of evidence to test construct validity, create a rich theoretical framework from previous literature to ensure both internal and external validity and build case study protocols and databases for reliability.

VI. Results - SPM and SA Collaboration Patterns

All case companies have similar approaches to SPM and SA. To begin, all claim to practice agile development, which prescribes a distinct (management) approach. The architects in particular have loosely defined roles, which they grew into as the company and their seniority grew. They are the technical lead of a development team with the additional responsibility to ensure sound architectural decisions. As [12] puts it, the architect is considered "first among equals", leading the development team and communicating with relevant stakeholders such as the product manager. Due to the organizations' agile approach, architects only make ADDs before the start of a project for architecturally relevant requirements while ADDs for regular requirements are made on an ad-hoc basis. Identifying which requirements are architecturally relevant is done during requirements refinement, which each case companies conducts to some extent. Subsection VI-B further details this collaborative effort.

A. Generic Findings

None of the respondents declared that transferring product context is an important part of SPM and SA collaboration. Yet, our case study architects possess deep contextual knowledge due to two circumstances. As the product grows, the architect's contextual knowledge organically grows along. When an architect works on the same product for a long time, he is bound to establish an accurate contextual model. Furthermore, the architects at 3 companies were part of the product's initial development team and thus had intimate knowledge of the domain, while the product manager role was introduced only after product success. Respondents argue that these factors substitute the need for contextual clarification. Architects do rely on SPM to communicate with customers and users. Product managers function as intermediaries for external stakeholder requests that might concern the architect. Support and sales first report to the product manager, who decides whether to escalate the issue to the architect. Any resulting information or actions traverse the same paths.

Contrary to what previous literature prescribes [1], product managers at our case study companies typically also assume project management responsibilities. While each SPO gathers, identifies, organizes, and prioritizes requirements to create new release definitions, none had identical approaches. Despite this difference, all product managers independently identify and organize requirements. The responsibility and stakeholders involved during requirements gathering and prioritization diverge. Three case companies have formalized meetings where the product manager gathers requirements from the architect. The remaining architects transfer requirements on an incidental basis. One product manager conducts prioritization independently, only consulting the architect when necessary. At three other SPOs the product manager relies on the architect's

input to make informed requirement decisions. The extent of this reliance depends on the product manager's technical product knowledge and capabilities. The following paraphrase succinctly illustrates the typical difference between product managers with an engineering and marketing background:

> Alice and I are product managers at FinComp. Alice has an engineering background, I have a lot of experience in professional services. Before the start of a sprint, Alice works out a first level of architecture and validates it with the architect. I told the architect not to even come to me with technical questions because I cannot provide answers. Alice can challenge architectures he comes up with, I cannot and thus rely 100% on the architect to advice me on technical requirements.

One SPO has a special committee to collaboratively make decisions on requirements. Although the resulting broad stakeholder support is beneficial, the organization loses decisiveness and agility due to extra process overhead. The product manager and architect proposed a solution:

> "We'd like a portion of the release to fully be our responsibility so we can autonomously decide to implement some product requirements regardless of current customer requests"

B. Important Overlapping SPM & SA Processes

Product managers and architects exchange information during two activities: requirements *gathering* and *refinement*. Gathering requirements from a variety of different stakeholders is crucial as a product manager [21]. Of all the internal stakeholders that contribute to the product manager's requirements gathering process, the architect is unique. He is the only actor with the capability to identify technical debt and formulate requirements to solve critical deficiencies. One respondent that transitioned from engineering into SPM clarifies the importance of formally requesting the architect for input:

> "Gathering requirements from the architect is our most important collaborative process. Due to my functional focus I no longer notice where code quality is degrading and overdue maintenance is growing."

The quote above illustrates that even product managers with a (strong) engineering background are unable to maintain in-depth technical knowledge of the product(s) they manage. This characterizes the first interdependence of SPM and SA: the product manager is responsible for scheduling requirements including technical maintenance, while the architect has the domain knowledge to identify which product aspects need maintenance. To this end, three of our case companies have scheduled meetings to discuss technical maintenance. The architect presents which product sections require attention and why, the product manager shares his position from a business perspective and together they decide what issues to invest in. During these meetings information exchange is verbal, at times supported by simple informal models that happen to resemble the Functional Architecture Model by [41]. Depending on the technical competence of the product manager, both spend less or more energy on this communicative effort. At another case company, technical maintenance is a standard part of each sprint and the architect's responsibility. While this permits more effectiveness and autonomous decisiveness for the architect, the risk of focusing resources on the wrong issues increases.

Each of the case companies conducts requirement refinement, though none have adapted [31]'s guidelines. Advanced concepts such as a product management backlog and alternating sprints with development have not been formally integrated into the organizations. Instead, SPM refines requirements on a case by case basis into high-level definitions. Next, during a formalized (bi-)weekly meeting, product management refines requirements with the development team and architect in an agile, iterative manner. The product manager attends these meetings to coordinate and support the development team. He answers questions development poses, validates their interpretation of his work and might even lead the meeting. Different topics come up, at times including requirement validation and dependency linking; sub-activities of requirements identification and organizing. Two product managers note that it is imperative to convey *why* they want to develop a feature. This enables developers to autonomously answer questions, preventing additional information requests during development and/or incorrectly implemented features. Requirements go through as many refinement iterations as necessary until the requirement definition has sufficient detail for development to start developing in the next sprint. Effectively, SPM and SA are going through the Twin Peaks stages until they hash out the lowest level of detail from the requirements perspective. Due to the frequency of this collaboration and importance to creating a winning product and delivering value to customers, we claim that requirement refinement is the most important process during which product managers and architects collaborate and exchange information.

C. Instrumentations Facilitating SA & SPM Information Exchange

All respondents recognize that they use a variety of tools, but note that few are applicable to this research context. For example, each SPO uses a tool to track features such as Jira, OnTime or Pivotal Tracker. These tools are central to their development activities, primarily benefiting short term project management instead of the more conceptual information exchange between an architect and product manager.

Just one of the case companies claims to use a tool specifically to facilitate information exchange between technical and less technical roles: Enterprise Architect (EA). Although EA has a variety of features, this company primarily used its modeling capabilities to create both technical and functional models of their product which are connected by a data model. The product manager discusses their EA experiences:

> "EA improves [people] alignment and [communication] clarity. Recording ideas in models well enables us to discourse efficiently; explaining how, why and what you want becomes comprehensible. In turn, translating functionality and business requirements into technical requirements is now significantly more logical. Furthermore, because all stakeholders operate around the different models, alignment occurs far earlier in the lifecycle."

Despite these advantages, respondents from the other four case companies explicitly stressed their aversion to static documentation tools. Although none object to modeling itself, they warn against meticulously creating models for company-wide usage. Because, despite their meticulous effort, modeling is a human activity that *will* produce errors. One respondent stated: "No documentation is better than outdated and thus wrong documentation", to which his co-interviewee, a product manager with limited technical background, added the following nuance: "If it's not a living document, it does not serve a purpose. If it's got a lifespan of X amount of time, don't invest in it. If you know it's a living document that you're going to refer to for the rest of your life, then nobody has an issue updating it.". Yet, neither could think of a document within their SPO with this kind of urgency.

Instead, these respondents create rough models, (functional, data, flow, etc) on a whiteboard when necessary. Aside from always being up to date, this approach permits hands-on collaboration on the whiteboard and communicative flexibility in what aspects of the model you draw, directing focus to only those parts that matter. Often, these models resemble the Functional Architecture Modeling technique [41], consisting of high-level elements that exchange high-level data objects. When participants deem a specific architectural approach is desirable, someone photographs the drawing and adds it to the requirement specification. The drawing then provides high-level guidance without becoming a fully thought-out implementation plan.

VII. Discussion

Case study analysis shows that all of the SPOs have some form of SPM and SA interdependence, typically addressed at a (bi-)weekly meeting. Although the formalization degree of these meetings varies, requirements are a common central topic. We found direct indications for three out of four collaborative activities presented in subsection IV: (1) requirements gathering, (2) prioritization and (3) refinement. Much to our surprise, respondents indicate that architects do not need (4) product context clarification from product managers to identify important QAs and formulate correct architectural approaches.

The case study SPOs avoid having to communicate product context to new architects by not hiring external recruits for architectural roles. Instead, they promote software engineers whose contextual knowledge already is close to on par with the product manager. Furthermore, the architect receives sufficient contemporary contextual information in an implicit manner during his involvement with the product. However, we believe explicit context exchange is beneficial for the architect's understanding in the long run. One architect clarifies: "We used to be our own customers. Nowadays we are out of touch with what it's like to work in the domain. Because we haven't done the work in 15 years and the field has evolved drastically, we need to test our assumptions with customers."

All other requirement related activities are part of the collaboration: product managers request SA input for requirements prioritization, the architect has a unique contribution for requirements gathering and together they refine requirements into sufficient detail for development. Additionally, during refinement different topics come up including requirement validation and dependency linking. Through these two sub-activities, they collaborate on requirements identification and organization indirectly as well. Finally, the product manager acts as an intermediary between the architect and customer requests and requirements.

Requirements gathering and refinement are particularly interesting due to the complex, reciprocal information exchange that takes place. When a product manager gathers requirements from internal stakeholders, the goal is to include the perspective of all relevant stakeholders [45] to gain insight into what issues are currently important within the organization. Unfortunately, there is no specific literature describing how product managers gather requirements, let alone how specific stakeholders take part in this activity. Without this information, formulating approaches or designing instrumentations to support SPM and SA alignment is impossible. Our case studies show that architects contribute a unique class of requirements because the product manager is unable to reliably identify technical debt himself. Thus the architect attempts to clarify the product's weak aspects in formalized meetings. However, he can only do so by conveying technical details that are conceptually out of scope for the product manager.

The extent to which our product management respondents participate in requirements refinement is out of scope in a similar manner. In theory, development conducts requirement refinement to add sufficient detail to underspecified requirements, enabling each team member to fulfill any requirement in a later stage. However, four of the product managers participate in requirements refinement sessions. In their experience, the independent, document-driven approach for refinement described in [31] generates too many requests for more details. Instead, architects refine requirements collaboratively with product managers to resolve impediments as early and effectively as possible.

These findings enable us to formulate an answer to our main research question. Software product managers and software architects collaborate in three ways: they exchange information to gather requirements, collectively prioritize requirements and iteratively refine requirements selected for a release. These three activities have large, direct influence on product success. After all, gathering, selecting and refining the *right* requirements in the *right* manner is a prerequisite to delivering the *right* value to customers.

Consequentially, aligning involved stakeholders in these tasks is imperative to creating a winning product. Figure 4 presented the reciprocal contributions between SPM and SA, demonstrating that the key to alignment is facilitating their different forms of information exchange by creating a shared understanding of the product. Models that both stakeholders are able to reason about and discuss have the potential to attain this goal. However, of our respondents, only one SPO utilizes formalized technical and data models structured in a company-wide standardized manner. Although their experience with the tool is positive, all other respondents note one critical drawback: human-made models are outdated, wrong or both and using them will lead to corresponding decisions.

These skeptical respondents rely solely on the most basic communicative tool, drawing on a whiteboard on an incidental basis. [46] uncovered similar results, noting that architects primarily utilize whiteboard sketches to communicate architecture models to secondary stakeholders. While they claim the benefits of these transient models are great, they forgo SA's analytical and educational merits. Moreover, we question the accuracy and timeliness of these high-level model sketches in practice. Especially considering that as product complexity grows and subsequently the number of people involved in development, accurately depicting an SA from memory becomes exponentially more difficult. This presents us with a conundrum: if neither formal nor incidentally drawn models are applicable for facilitating SPM and SA alignment, what approach should an SPO take?

VIII. Proposed Solution

SPOs face a dilemma in their efforts to align SPM and SA. Our case studies show that SPOs have two approaches to documenting a product: multiple, formalized models or no models at all. Over time, the former leads to outdated, inaccurate models that harm the organization, while the latter forgoes all benefits of models by relying on stakeholders' communicative competence to exchanging the right information. We require an approach that utilizes models understandable for both stakeholders and ensures their timeliness and accuracy.

Combining the literature review and results from our case studies, we identify four architectural model uses relevant for product software: (1) *incidental models* which stakeholders draw when the need presents itself during meetings, (2) *informal models* that record incidental models for later reference if necessary, (3) *discussion models* depict the product on a high (functional) level to enable technical stakeholders to exchange information with non-technical stakeholders and (4) *structured models* take a formalized approach at depicting architecture such as data models for documentation purposes, possibly captured in tooling. Of these, discussion models (DMs) are particularly relevant for our purposes. Anyone can create a simple DM by using requirements as guidance and DMs can have multiple degrees of detail for different stakeholders. These characteristics adhere to the requirements argued by [47] for aligning requirements with architecture:

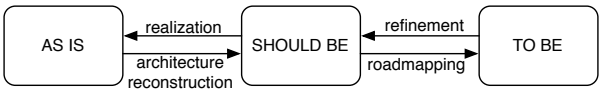


Fig. 5. Accurate Architectural Models Approach

requirements are decomposed into fine grained specifications and organized in a similar manner to the system's architecture.

Unfortunately however, DMs become outdated and inaccurate as well. While [40] prescribes continuously monitoring the architecture and adjusting the FAF when it is no longer accurate, the following scenario happens in practice: over time the discussion model (DM) starts to diverge from the implemented situation. The architect, however, is confident the DM is accurate enough for the task at hand and chooses not to update it or forgets to do so after implementation. Meanwhile, the product manager re-uses the inaccurate model for new requirements. Three iterations later the DM diverges from reality to such an extent that it is useless and both stakeholders complain when they have to create a new one. Concisely, human error and/or negligence causes DMs to become outdated, wrong or both as well.

While many different models exist, SPOs lack a structured approach to prevent model divergence from the implemented situation. To this end we propose the Accurate Architectural Models Approach (AAMA) that engages both the product manager and architect to ensure ongoing accuracy and timeliness. AAMA consists of three co-existing model instantiations: a current technical truth, a future design and a combination of the two. This division is similar to ones used in architecture compliance and auditing [48], [49]. To work with these models, stakeholders follow four distinct steps as shown in Figure 5.

- **AS IS** The product's current situation as seen from the architect's technical perspective.
- **SHOULD BE** The product's current design, consists of the current situation and all pending design changes based on new requirements.
- **TO BE** A future design of the product based on new requirements to be included in a release.

This paragraph presents a scenario introducing how SPOs can apply AAMA to improve SPM and SA alignment. As a product matures, the product manager and architect recognize the need to approach their collaboration in a more sophisticated manner and decide to apply AAMA to their discussion models. Together they create an *origin model* based on the product's current situation to facilitate their information exchange. On this originating moment, the AS IS and SHOULD BE are identical and both stakeholders agree to apply AAMA by adhering to the following four steps.

1) **Roadmapping** New requirements prompt the product manager to use the origin model to create a TO BE.
2) **Refinement** In the second step, both stakeholders refine the TO BE to create a SHOULD BE.

3) **Realization** The architect consults the SHOULD BE to realize the new requirements.
4) **Architecture Reconstruction** After realization, a new AS IS situation materializes which diverges from the SHOULD BE, prompting the architect to conduct architecture reconstruction to warrant the SHOULD BE's accuracy.

Finally, the cycle restarts when the product manager uses the newly updated and accurate SHOULD BE as input.

Applying AAMA has three consequences for the models position in the organization. (1) Frequent reviews and updates makes the DM a *living model*, preventing outdated models and human made errors. (2) By ensuring that architects as well as product managers examine and update the DM on a frequent basis, both have a deep understanding of the recent changes to the product. (3) Having a high architectural view available on demand enables all stakeholders to create incidental models of strictly those product sections relevant to the discussion. In turn, these consequences have three positive influences on the reciprocal contributions of SPM and SA:

1) The common view and mutual deep understanding assists the frequent communication on product requirements and requirements feedback.
2) The architect transfers the right architectural product knowledge more effectively and efficiently because he knows the extent of the product manager's architectural knowledge.
3) AAMA's iterative TO BE DM refinement aligns with the Twin Peaks model. The product manager expresses requirements in the TO BE DM, which the architect examines for implementation and requests further refinement details from the product manager. The resulting resulting release definition is more complete and unambiguous, contributing to the architect's technical product strategy.

Although AAMA prevents cumulative DM divergence from the current implementation, one major shortcoming remains in the product software context. During implementation, the product manager is already processing new, continuously incoming requirements into a new TO BE DM. The moment the SHOULD BE DM is updated, the product manager has already produced a new TO BE DM, possibly creating a divergence. This divergence will result in either the product manager having to alter his DM or the architect to work with an inaccurate specification, reducing the DM's utility.

AAMA is a giant leap forwards, but requires further improvements to achieve continuous model accuracy. Some respondents wish to automatically generate high-level architecture views from data supplied by instrumentations they already use. A sentiment echoed in literature: "the most useful forms of documentation are views of the software that can be automatically generated" [50]. Automatically generating a DM from source code and subsequently connecting all elements to requirements from the requirement database is an exciting concept. However, automatic architecture reconstruction to date has only a few incomplete successes for specific uses cases [51], [52]. The accuracy and relevance of an instrument that automatically generates architectural views is doubtful and the prospect of having the architect adjust the entire view manually is considerably less exciting.

Instead, future design science research will use this qualitative study's results to formulate an automated prototype. It will unite the three different DM instantiations into one shared, digital model to prevent divergence at any stage. After prototype improvement interviews with practitioners, we plan to conduct a survey to determine Dutch product managers' and software architects' usage intention towards AAMA.

IX. Conclusion

The start of this paper asks *"how do software product managers and software architects collaborate?"*. By means of five elaborate case studies we uncover two critical processes where product managers and architects exchange information: requirements gathering and requirements refinement. These activities require reciprocal information exchange on a high technical level to formulate and refine the right requirements in the right way. The successful execution of these processes is a critical driver to product success. Respondents, however, rely on primitive methods and lack a structured approach that warrants SPM and SA alignment. We believe SPOs can improve their SPM and SA alignment by adopting discussion models and propose the Accurate Architectural Models Approach (AAMA) to prevent architectural model divergence.

In future research, we aim to further elaborate and formalize the SPM and SA interplay in order to benefit their alignment and resolve discrepancies between current literature and the findings in this paper. For instance, we intend to determine whether requirements refinement is truly out of scope for product managers, internal promotions always fill architectural vacancies and validate AAMA's potential benefits compared to updating models on an incidental basis. Furthermore, designing and validating an instrumentation to facilitate information exchange further establishes the research domain. Finally, a longitudinal study measuring the effect of applying AAMA and/or facilitating instrumentations will validate the resulting artifacts and impact of our efforts.

References

[1] C. Ebert, "The Impacts of Software Product Management," *Journal of Systems and Software*, vol. 6, no. 80, pp. 850–861, 2007.
[2] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proceedings of WICSA '05*, 2005, pp. 109–120.
[3] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little, *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002.
[4] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115–119, Mar. 2001.
[5] P. T. Ward and S. J. Mellor, *Structured Development for Real-Time Systems: Vol. I: Introduction and Tools*. Prentice Hall, 1986.
[6] M. Khurum and T. Gorschek, "A method for alignment evaluation of product strategies among stakeholders (mass) in software intensive product development," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 7, pp. 494–516, 2011.
[7] B. Regnell, R. Berntsson Svensson, and T. Olsson, "Supporting roadmapping of quality requirements," *Software, IEEE*, vol. 25, no. 2, pp. 42–47, 2008.

[8] K. Wnuk, B. Regnell, and C. Schrewelius, "Architecting and coordinating thousands of requirements an industrial case study," in *Requirements Engineering: Foundation for Software Quality*, ser. LNCS. Springer, 2009, vol. 5512, pp. 118–123.

[9] D. Condon, *Software Product Management*. Boston, Massachusetts: Aspatore Books, 2002.

[10] A. Helferich, K. Schmid, and G. Herzwurm, "Product management for software product lines: An unsolved problem?" *Communications of the {ACM}*, vol. 49, no. 12, pp. 66–67, Dec. 2006.

[11] M. W. Maier, D. Emery, and R. Hilliard, "Software architecture: Introducing ieee standard 1471," *Computer*, vol. 34, no. 4, pp. 107–109, Apr 2001.

[12] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2010.

[13] J. ISO, "IEC 25010: 2011: Systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models," *International Organization for Standardization*, 2011.

[14] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti, "Relating software requirements and architectures using problem frames," in *IEEE International Requirements Engineering Conference (RE'02)*, 2002, pp. 137–144.

[15] P. Grünbacher, A. Egyed, and N. Medvidovic, "Reconciling software requirements and architectures with intermediate models," *Software and Systems Modeling*, vol. 3, no. 3, pp. 235–253, 2004.

[16] R. Chitchyan, M. Pinto, A. Rashid, and L. Fuentes, "Compass: Composition-centric mapping of aspectual requirements to architecture," in *Transactions on Aspect-Oriented Software Development IV*, ser. LNCS. Springer, 2007, vol. 4640, pp. 3–53.

[17] K. M. van Hee, J. J. A. Keiren, R. D. J. Post, N. Sidorova, and J. M. E. M. van der Werf, "Designing case handling systems," in *Transactions on Petri Nets and Other Models of Concurrency I*. Springer, Berlin, 2008, vol. 5100, pp. 119 – 133.

[18] P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, and I. Mistrk, *Relating Software Requirements and Architectures*, 1st ed. Springer, 2011.

[19] H. Vogl, K. Lehner, P. Grunbacher, and A. Egyed, "Reconciling requirements and architectures with the cbsp approach in an iphone app project," in *Requirements Engineering Conference (RE), 2011 19th IEEE International*, Aug 2011, pp. 273–278.

[20] C. Chen, D. Shao, and D. Perry, "An exploratory case study using cbsp and archium," in *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale and Design Intent*. IEEE, May 2007, pp. 3–3.

[21] W. Bekkers, I. van de Weerd, M. Spruit, and S. Brinkkemper, "A Framework for Process Improvement in Software Product Management," in *Proceedings of EuroSPI*, 2010a, pp. 1–12.

[22] C. Ebert, "Software Product Management," *Crosstalk*, vol. 22, no. 1, pp. 15–19, 2009.

[23] L. Xu and S. Brinkkemper, "Concepts of Product Software," in *European Journal of Information Systems*, vol. 16, no. 5, 2007, pp. 531–541.

[24] A. Maglyas, U. Nikula, and K. Smolander, "What do practitioners mean when they talk about product management?" in *20th International Requirements Engineering Conference (RE)*, Sept 2012, pp. 261–266.

[25] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, and L. Bijlsma, "On the creation of a reference framework for software product management: Validation and tool support," in *International Workshop on Software Product Management*, 2006, pp. 3–12.

[26] L. Gorchels, *The Product Manager's Handbook: The Complete Product Management Resource (2nd edition)*, 2nd ed. Columbus, OH, USA: McGraw-Hill, 2000.

[27] C. Ebert and S. Brinkkemper, "Software product management an industry evaluation," *Journal of Systems and Software*, no. 0, pp. –, 2014.

[28] S. Fricker, "Software product management," in *Software for People*, ser. Management for Professionals. Springer, 2012, pp. 53–81.

[29] A. Maglyas, U. Nikula, and K. Smolander, "What do we know about software product management? - a systematic mapping study," in *Software Product Management (IWSPM), 2011 Fifth International Workshop on*, Aug 2011, pp. 26–35.

[30] H.-B. Kittlaus and P. N. Clough, *Software Product Management and Pricing: Key Success Factors for Software Organizations*. Berlin, Germany: Springer, 2009.

[31] K. Vlaanderen, S. Jansen, S. Brinkkemper, and E. Jaspers, "The agile requirements refinery: Applying {SCRUM} principles to software product management," *Information and Software Technology*, vol. 53, no. 1, pp. 58 – 70, 2011.

[32] A. M. Davis, *Just Enough Requirements Management: Where Software Development Meets Marketing*. New York, NY, USA: Dorset House Publishing Co., Inc., 2005.

[33] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 35–46.

[34] C. Potts, "Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software," in *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, 1995.

[35] L. Karlsson, sa G. Dahlstedt, B. Regnell, J. N. och Dag, and A. Persson, "Requirements engineering challenges in market-driven software development an interview study with practitioners," *Information and Software Technology*, vol. 49, no. 6, pp. 588 – 604, 2007.

[36] P. Sawyer, I. Sommerville, and G. Kotonya, "Improving market-driven re processes," in *Proceedings of the International Conference on Product Focused Software Process Improvement*, vol. 195, Oulu, Finland, 1999, pp. 222–236.

[37] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Software*, vol. 14, no. 5, pp. 67–74, 1997.

[38] J. Natt och Dag, B. Regnell, V. Gervasi, and S. Brinkkemper, "A linguistic-engineering approach to large-scale requirements management," *Software, IEEE*, vol. 22, no. 1, pp. 32–39, 2005.

[39] J. Kabbedijk, S. Brinkkemper, S. Jansen, and B. van der Veldt, "Customer involvement in requirements management: Lessons from mass market software development," in *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, 2009, pp. 281–286.

[40] T. Salfischberger, I. van de Weerd, and S. Brinkkemper, "The functional architecture framework for organizing high volume requirements management," in *Software Product Management (IWSPM), 2011 Fifth International Workshop on*, Aug 2011, pp. 17–25.

[41] S. Brinkkemper and S. Pachidi, "Functional architecture modeling for the software product industry," in *Software Architecture*, ser. LNCS. Springer, 2010, vol. 6285, pp. 198–213.

[42] P. Corbetta, *Social Research: Theory, Methods and Techniques*. London: Sage Publications, 2003.

[43] A. Kajornboon, "Using Interviews as Research Instruments," *E-Journal for Research Teachers*, vol. 2, no. 1, 2005.

[44] R. K. Yin, *Case Study Research - Design and Methods*. SAGE Publications, 2009.

[45] W. Bekkers, M. Spruit, I. van de Weerd, R. van Vliet, and A. Mahieu, "A situational assessment method for software product management," in *Proceedings of ECIS*, Pretoria, South-Africa, 2010b, pp. 22–34.

[46] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: How and why software developers use drawings," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '07. New York, NY, USA: ACM, 2007, pp. 557–566.

[47] M. W. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. P. E. Heimdahl, and S. Rayadurgam, "Your "what" is my "how": Iteration and hierarchy in system design," *IEEE Software*, vol. 30, no. 2, pp. 54–60, 2013.

[48] R. Koschke and D. Simon, "Hierarchical reflexion models," in *Proceedings of the 10th Working Conference on Reverse Engineering*, ser. WCRE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 36–.

[49] W. van der Aalst, K. van Hee, J. M. van der Werf, A. Kumar, and M. Verdonk, "Conceptual model for online auditing," *Decision Support Systems*, vol. 50, no. 3, pp. 636 – 647, 2011.

[50] M. Mirakhorli and J. Cleland-Huang, "Traversing the twin peaks," *IEEE Software*, vol. 30, no. 2, pp. 30–36, 2013.

[51] F. Schmidt, S. MacDonell, and A. Connor, "An automatic architecture reconstruction and refactoring framework," in *Software Engineering Research, Management and Applications 2011*, ser. SCI, R. Lee, Ed. Springer, 2012, vol. 377, pp. 95–111.

[52] F. A. Fontana and M. Zanoni, "A tool for design pattern detection and software architecture reconstruction," *Information Sciences*, vol. 181, no. 7, pp. 1306 – 1324, 2011.