Architectural assumptions and their management in software development

Yang, Chen

Chapter 1 Introduction

This thesis studies architectural assumptions and their management in software development. Section 1.1, 1.2, and 1.3 introduce background information about this topic: assumptions in software development, software architecture and architectural knowledge, and architectural assumptions and their management. Problem statement and research design are discussed in Section 1.4 and 1.5 respectively. Section 1.6 provides an overview of the thesis.

1.1 Assumptions in software development

During software development, there can be many uncertain things. However, in order to meet the project business goals (e.g., schedule and deadlines), stakeholders have to work in the presence of such uncertainties; these uncertainties can lead to assumptions. For example, uncertainty regarding the release date of a specific technology to be used in a system may lead to making an assumption about that release date. In this thesis, we advocate treating uncertainty and assumption as two different but related concepts: one way to deal with uncertainties is to make implicit or explicit assumptions, but not all uncertainties lead to assumptions.

An assumption is defined as "*a thing that is accepted as true or as certain to happen, without proof*"[1] or as "*a fact or statement taken for granted*"[2]. Accordingly, we define software assumptions as *software development knowledge taken for granted or accepted as true without evidence*. According to the work of Alavi and Leidner [193], as well as the edited book by Aurum *et al.* [194], software development knowledge represents personalized information related to facts, procedures, concepts, interpretations, ideas, observations, and judgments in software development. This definition of assumption emphasizes the characteristic of uncertainty in software development: stakeholders believe but cannot know for sure the importance, impact, suitability, applicability, correctness, etc. of software development knowledge. For example, consider a project manager assuming that "*the skills and capacities of the software engineers in the development team are sufficient for this project*". In this statement, the project manager is not 100% sure about the sufficiency of the skills and capacities of the software engineers. As another example, a developer may assume that "*changing Component A would not impact the other components in the system*". In this statement, the developer is not completely sure about the impact of changing Component A.

[1] http://www.oxforddictionaries.com/definition/english/assumption

[2] http://www.merriam-webster.com/dictionary/assumption

In addition to being software development knowledge, software assumptions are also a type of artifact. As defined by Kroll and Kruchten [35]: "*An artifact is a piece of information that is produced, modified, or used by a process*". Since assumptions are produced, modified, and used during software development, we advocate treating assumptions as a type of software artifact, similarly to requirements, design decisions, etc.

Assumptions in the field of software development constitute a broad topic: different types of assumptions (e.g., requirement assumptions [5], architectural assumptions [4], and software construction assumptions [6]) have been extensively discussed. Accordingly, different stakeholders such as designers, requirements engineers, developers, and testers frequently make assumptions in their work [7].

Many researchers have pointed out the importance of assumptions and their management in software development, as various problems in software development can be traced to not well-managed assumptions [7]. Five examples of such problems are provided as follows. First, Corbató [8] mentioned in his ACM Turing Award lecture that "*design bugs are often subtle and occur by evolution with early assumptions being forgotten as new features or uses are added to systems.*" Second, Garlan *et al.* [9] pointed out that incompatible assumptions in software architecture could lead to architectural mismatch (e.g., mismatch between components or connectors). Subsequently this may lead to design violations and low architecture quality. Third, stakeholders may misunderstand assumptions, because the assumption concept is rather subjective [4]; this can further lead to misunderstandings of other types of software artifacts that are related to such assumptions. For example, stakeholders may misunderstand architectural design decisions, because they are not aware of the assumptions behind the decisions. Fourth, Steingruebl and Peterson [11] argued that undocumented software assumptions could lead to software failures. As an example, assuming a system will run as a single-user standalone system, and therefore, there is no need to consider security concerns such as cross-site scripting or system permission mechanisms. Leaving such an assumption undocumented could further lead to security problems, especially when the context changes (e.g., the system will be deployed directly on the Internet). Fifth, Bazaz *et al.* [12] defined vulnerability as "*a state of the system from which it is possible to transition to an incorrect system state*", and pointed out that the violation of assumptions about system resources might cause the system to be vulnerable (e.g., memory exploits and I/O system exploits).

1.2 Software architecture and architectural knowledge

Software Architecture represents "*the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*" [1]. Every system has an architecture [2]. Software architecture

acts as a high-level design and as a means of performing complicated trade-offs among functional, non-functional, and business requirements [2].

The topic of software architecture has a long history in both academia and industry [22]. It shifted from the concept of system structure and behavior (i.e. components interacting through connectors), to the concept of architectural knowledge (software architecture is comprised of a set of design decisions and design) [23]. The latter concept goes beyond viewing architecture as merely the end result; it also focuses on the process that stakeholders follow to reach that end result, i.e., architectural knowledge management [24][25].

The importance of architectural knowledge and its management (e.g., documentation, sharing, and reuse) has been emphasized by both researchers and practitioners over the past years [23]. The benefits of managing architectural knowledge are various [23], such as: (1) reducing knowledge vaporization in software development; (2) mitigating misunderstandings and ineffective communications between stakeholders; (3) facilitating a better understanding of the architecture as well as the whole system within a project team; and (4) helping system analysis (e.g., impact analysis of design decisions).

1.3 Architectural assumptions and their management

According to the aforementioned definition of software assumption (see Section 1.1), we define architectural assumptions as architectural knowledge taken for granted or accepted as true without evidence. As an example, a stakeholder may assume that "*the number of users (visitors) of the system would be around 1 million per day*". When the uncertainty of an architectural assumption is eliminated, the assumption can be removed or transformed to another type of software artifact (in the case of the previous example the assumption will become a requirement when the number of users (visitors) of the system turns out to be 1 million per day as initially thought). Like other types of assumptions, architectural assumptions have a set of characteristics, described as follows:

(1) **Subjective**. Many researchers and practitioners pointed out the subjective nature of assumptions in software development (i.e., whether a piece of information is an assumption or not, is rather subjective). This is the major reason that stakeholders may have a different understanding of the assumption concept. As an example, Roeller *et al.* [4] mentioned: "*From one perspective or stakeholder, we may denote something as an assumption, while that same thing may be seen as a design decision from another perspective.*"

(2) **Dynamic**. Assumptions have a dynamic nature, i.e., they can evolve over time [7]. For example, during software development, a valid assumption can turn out to be invalid or vice versa, or an assumption can transform to another type of software artifact or vice versa.

(3) **Context dependent**. Assumptions are context dependent [10]. For example, the same assumption could be valid in one project, and invalid in another project because the context changes; or an assumption in one project is not an assumption in another project.
(4) **Intertwined with certain types of artifacts**. Assumptions are not independent in software development, but intertwined with many types of software artifacts. For example, when managing assumptions in software design (e.g., [34]), assumptions are commonly related to requirements, design decisions, components, etc.

In this thesis, of all the different types of assumptions in software development, we focus on architectural assumptions. The reasons are: (1) Architectural assumptions are an important type of architectural knowledge [4]. (2) Assumptions should be managed from the early phases of software development (i.e., requirements engineering and architecture design) [13]. As evidenced in many studies (e.g., [4][9][13]), managing architectural assumptions is of significant importance in both architecting and software development. (3) Many problems are caused by not-well managed architectural assumptions, such as architectural mismatch [9].

1.4 Problem statement

Even though we found many studies (e.g., [28][29][30][31][32][33]) regarding assumptions and their management in software development, the majority of the related work does not deal with architectural assumptions but other types of assumptions within different phases of software development. However, the proposed approaches, techniques, and tools for managing assumptions in other phases of software development may not be suitable for architectural assumption management. As an example, there are studies on the topic of assume-guarantee reasoning (e.g., [30][32]), which is a powerful approach for system verification, including modular, component, and program verification, and can support assumption management activities, such as Making, Description and Evaluation. Nevertheless, in other contexts of software development (e.g., making architecture design decisions), assume-guarantee reasoning may not be suitable [14].

There is also some related work that targets architectural assumptions and their management specifically (e.g., [4][9][13][27][34]). However, we see the following limitations in those works:

(1) There is no general architectural assumption management process proposed in literature, only approaches for individual activities. For example, Roeller *et al.* [4] focused on Architectural Assumption Recovery (e.g., how to recover architectural assumptions), while Recovery is only a single activity within architectural assumption management. The "big picture" of architectural assumption management in software

development is missing, resulting in a lack of systematic and efficient management of architectural assumptions in software development.

(2) Many approaches for architectural assumption management are considered resource-intensive, leading to a rather low return on investment. This is a key challenge in having those approaches adopted in industrial practice.

(3) There is a lack of clear and practical guidance for software development teams to apply the proposed approaches regarding architectural assumption management in current literature. Specifically, we see the following problems: (a) Some studies only suggest that an approach can be used in managing architectural assumptions, without further elaboration (e.g., how exactly such an approach can be used to manage architectural assumptions). (b) Different stakeholders have various concerns about architectural assumptions, but the existing approaches only address few of them, and the connections between such concerns and respective stakeholders are not clear; and (c) It is ambiguous which architectural assumptions concerns are addressed by the proposed approaches, and how they address the concerns.

(4) There is a lack of dedicated tools in architectural assumption management. For example, stakeholders usually use MS Word, Excel, etc. to document architectural assumptions. However, these general tools are not dedicated to Architectural Assumption Description, which leads to a number of problems (e.g., description becomes too resource-intensive and difficult to manage).

We see two reasons that cause the aforementioned problems. First, while treating architectural assumptions as first class entities is of significant importance in software development, it is not yet practiced at a large scale. Instead, architectural assumptions are usually mixed with other types of artifacts, or considered as, for instance, a force of another type of artifact. Second, due to the characteristics of assumptions, especially their subjective nature, many studies mention that it is difficult to draw a line between assumptions and other types of software artifacts. This hinders architectural assumption management in software development, as assumptions are mixed with other types of artifacts.

In this thesis, we address the core problem: **how can we provide a systematic approach to manage architectural assumptions?**

1.5 Research design

As defined by March and Smith [36], as well as Hevner *et al.* [37], design science is "*the design and validation of solution proposals to practical problems*". Wieringa [38] further defined design science as "*the design and investigation of artifacts in context*". Such artifacts that interact with a problem context are used to improve something in the specific context [38]. The design and investigation part can be mapped to

two types of problems [38], namely design problems (i.e., "*call for a change of the world*", e.g., how to improve things?) and knowledge questions (i.e., "*call for a change of our knowledge about the world*", e.g., what is the state of the art of things?).

As one problem can generate more problems, this makes the development of a design science project iterative. For example, when we design an artifact to address a design problem, we may need to ask certain knowledge questions regarding, for instance, the artifact itself or its problem context. Answering such knowledge questions can offer knowledge to help address the design problem. As another example, when answering a knowledge question, it can lead to one or several new design problems. Addressing such design problems can help to answer the original knowledge question.

Wieringa [38] developed an engineering cycle for design science, comprised of problem investigation, treatment design, treatment validation, treatment implementation, and implementation evaluation. As an example, a researcher can start from investigating a practical problem; design one or several solutions for the problem; evaluate the solutions; select one design to implement; evaluate the outcome of the implementation; and there could be a new iteration starting from the beginning within the engineering cycle.

There are several frameworks for design science. As an example shown in Fig. 1, Wieringa [15] refined the framework proposed by Hevner *et al.* [37]. Such a framework emphasizes not only designing artifacts to address a problem or evaluating a solution, but also problem investigation, namely paying attention to existing problems, goals, and outcomes [15]. Moreover, the structure of design science in this framework is comprised of two types of problems, i.e., design problems (also called as practical problems) and knowledge questions.

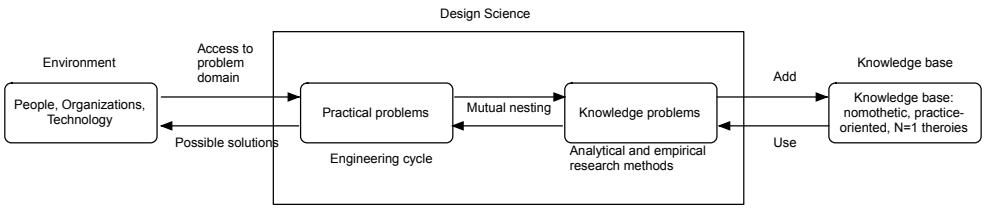

Fig. 1. Design science framework adapted from [37]

As another example shown in Fig. 2, Wieringa [38] further adapted his framework from [15]. The author used new terms, as well as new insights on the elements in the adapted framework. For instance, instead of classifying the knowledge base into three types in [15], the new framework calls it as knowledge context, while defines it in a more general way. However, the essence (e.g., key elements, such as design problems and knowledge questions) of the two frameworks is still similar.

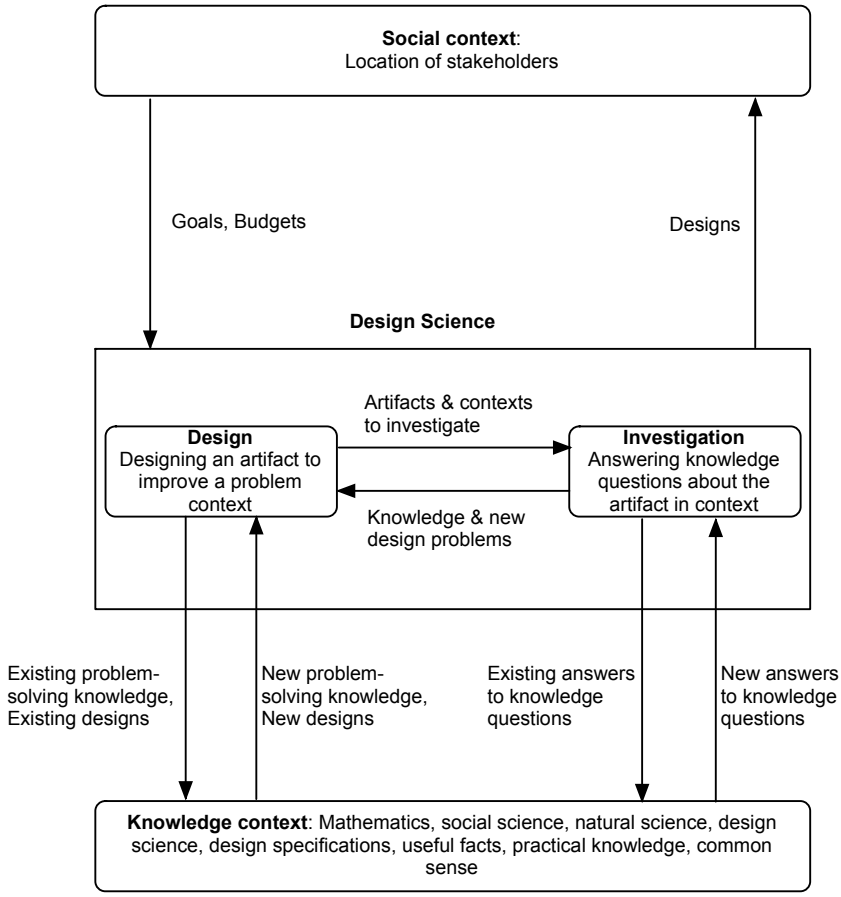

Fig. 2. Design science framework adapted from [15]

In the rest of this section, we first detail the design problems and knowledge questions of the research, and then provide the research methods used in the thesis.

1.5.1Design problems and knowledge questions

Fig. 3 shows the decomposition of the overall research problem into four design problems (i.e., DP1 – DP4) and four knowledge questions (i.e., KQ1 – KQ4).

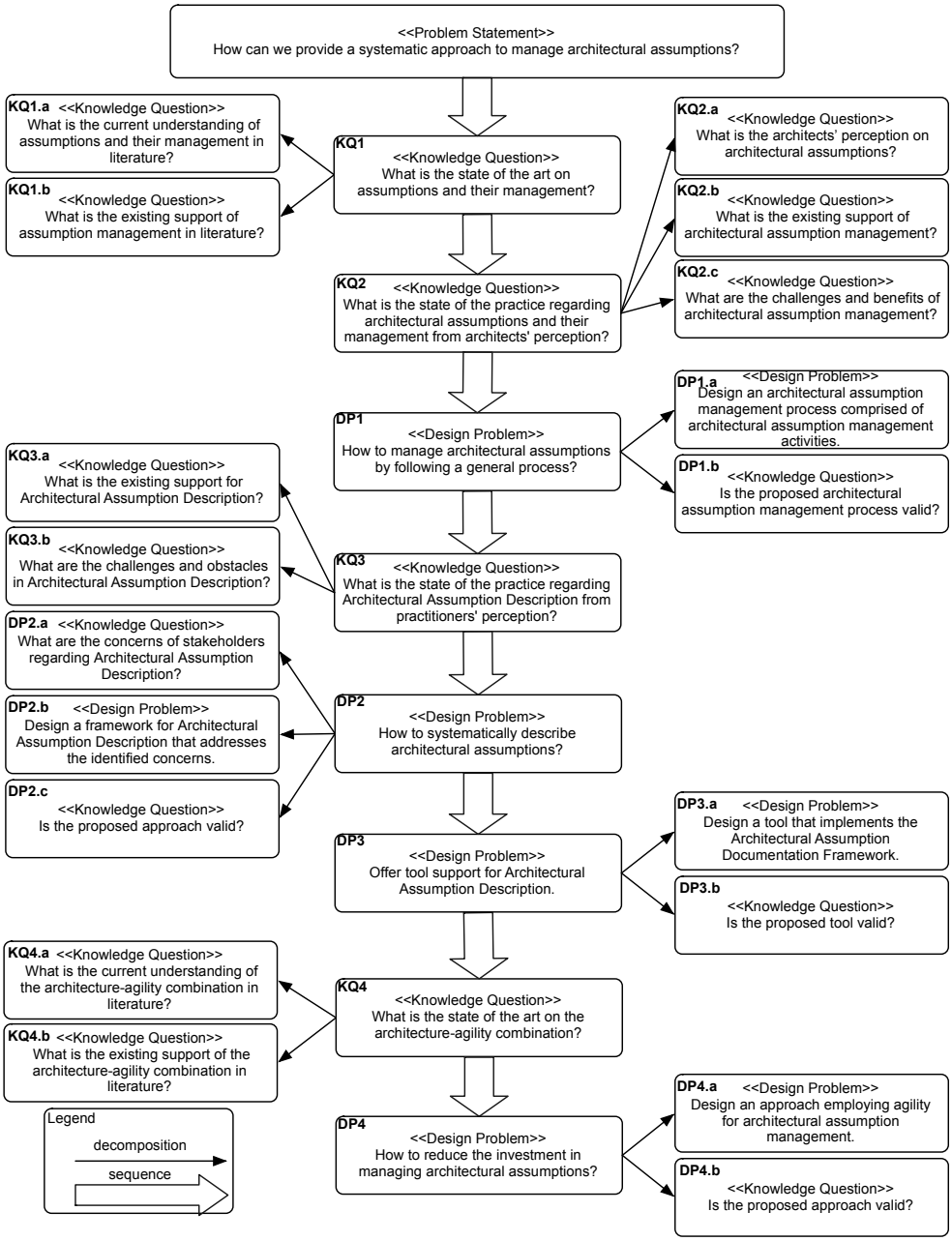

Fig. 3. Decomposition of the research problem

Before addressing the core problem stated in Section 1.4 (i.e., "*How can we provide a systematic approach to manage architectural assumptions?*"), there was a need

to analyze the research literature and understand the current state of the research regarding assumptions in general and their management in software development. Therefore, we came up with KQ1 (i.e., "*What is the state of the art on assumptions and their management?*"). KQ1 is further decomposed into two knowledge questions:

KQ1.a: "*What is the current understanding of assumptions and their management in literature?*" KQ1.a includes various aspects: definitions, classifications, and related software artifacts of assumptions; existing activities, related stakeholders, benefits, challenges, and lessons learned of assumption management; consequences caused by not well-managed assumptions.

KQ1.b: "*What is the existing support of assumption management in literature?*" KQ1.b concerns existing approaches and tools used for assumption management.

Besides the analysis of literature, it was of paramount importance to investigate the state of the practice regarding architectural assumptions and their management in industry. The major reason is that there could be a significant difference between academia and industry regarding the same topic, leading to different results. In this thesis, we strived to deal with real problems from industry, instead of academic problems. To this end, we formulated KQ2 (i.e., "*What is the state of the practice regarding architectural assumptions and their management from architects' perception?*"). In addition, while KQ1 regards assumptions in software development in general, the scope of KQ2 is narrowed down to architectural assumptions, which is the focus of this thesis. KQ2 is further decomposed into three knowledge questions:

KQ2.a: "*What is the architects' perception on architectural assumptions?*" KQ2.a studies the way architects perceive the term and concept of architectural assumption, examples and characteristics of architectural assumptions, and relations between architectural assumptions and other types of software artifacts.

KQ2.b: "*What is the existing support of architectural assumption management?*" KQ2.b aims to explore the existing architectural assumption management approaches and tools in industrial practice.

KQ2.c: "*What are the challenges and benefits of architectural assumption management?*" KQ2.c explores why architectural assumptions are usually not well managed in software development.

The results of KQ1 and KQ2 show that architectural assumption management is comprised of a set of architectural assumption management activities, but there is no general architectural assumption management process that can encompass these identified individual activities as a whole. Again, according to the results of KQ1 and KQ2, this is one of the major reasons that architectural assumptions are usually not systematically managed in software development. Therefore, we formulated DP1: "*How to manage architectural assumptions by following a general process?*" DP1 is further decomposed into a design problem and a knowledge question:

DP1.a: "*Design an architectural assumption management process comprised of architectural assumption management activities.* "

DP1.b: "*Is the proposed architectural assumption management process valid?*"

During the evaluation of the proposed architectural assumption management process, we found that it was of significant importance to provide dedicated approaches on individual activities of the process, especially Architectural Assumption Description, which is the most significant activity in managing architectural assumptions. As an example, it proves to be quite difficult to evaluate or maintain an architectural assumption if it is not described in a systematic way. However, before being able to propose a solution for Architectural Assumption Description, a more specific analysis was required of how this activity is performed in practice. To this end, we formulated KQ3 (i.e., "*What is the state of the practice regarding Architectural Assumption Description from practitioners' perception?*"). KQ3 is further decomposed into two knowledge questions:

KQ3.a: "*What is the existing support for Architectural Assumption Description?*" KQ3.a aims to further explore which approaches and tools were used in practice for Architectural Assumption Description.

KQ3.b: "*What are the challenges and obstacles in Architectural Assumption Description?*" KQ3.b gathers data regarding why architectural assumptions are usually not well documented in software development.

The results of KQ3 confirm that Architectural Assumption Description (or lack thereof) is a real problem in industry, and the existing approaches are not able to satisfy certain concerns from stakeholders in describing architectural assumptions in software development. This confirms the earlier results obtained from KQ1, KQ2. Therefore, we formulated DP2 as "*How to systematically describe architectural assumptions?*" DP2 is further decomposed into one design problem and two knowledge questions:

DP2.a: "*What are the concerns of stakeholders regarding Architectural Assumption Description*?" There is a lack of widely accepted understanding on the concerns of stakeholders that should be addressed when documenting architectural assumptions. A concern is any interest in a system related to stakeholders [1]. DP2.a aims to fill this gap.

DP2.b: "*Design a framework for Architectural Assumption Description that addresses the identified concerns.*"

DP2.c: "*Is the proposed approach valid?*"

During the evaluation of the framework, even though we found that the framework could benefit Architectural Assumption Description, the lack of tool support was a critical problem to adopt the framework in practice. To this end, we formulated DP3 (i.e., "*Offer tool support for Architectural Assumption Description*"), which is further decomposed into a design problem and a knowledge question:

DP3.a: "*Design a tool that implements the Architectural Assumption Documentation Framework.*"

DP3.b: "*Is the proposed tool valid?*"

During the evaluation of the tool, though we found it could improve architectural assumption management in several aspects (e.g., it proved indeed useful for Architectural Assumption Description), the effort required was still a key challenge of employing architectural assumption management in practice (also supported by the results of DP2). Therefore, there was a need to reduce the investment in managing architectural assumptions and consequently make it less resource-intensive. According to the literature (e.g., the Agile Manifesto [21]), agility aims at reducing the effort of traditional software development, and embracing changes with, for example, a set of agile practices. Integrating agility into architectural assumption management could be a promising way to address the aforementioned problem. As a first step, before trying to propose a specific solution, there was a need to understand the current state of the research regarding the combination of architecture in general and agility in software development. To this end, we formulated KQ4 (i.e., "*What is the state of the art on the architecture-agility combination?*"), which is further decomposed into two knowledge questions:

KQ4.a: "*What is the current understanding of the architecture-agility combination in literature?*" KQ4.a includes various aspects: benefits, costs, challenges, factors, and lessons learned of the architecture-agility combination.

KQ4.b: "*What is the existing support of the architecture-agility combination in literature?*" KQ4.b concerns existing approaches, practices, and tools used for the architecture-agility combination.

The results of KQ4 provide input on how to reduce the effort required in architectural assumption management by employing agility. For example, we learned which challenges should be dealt with, and what agile practices can be used when employing agility into architectural assumption management. Based on these results we could proceed to the following step, formulating DP4: "*How to reduce the investment in managing architectural assumptions?*" which is further decomposed into a design problem and a knowledge question:

DP4.a: "*Design an approach employing agility for architectural assumption management.*"

DP4.b: "*Is the proposed approach valid?*"

1.5.2 Research methods for the knowledge questions

In this thesis, we employed empirical research methods to address the knowledge questions mentioned in Section 1.5.1, namely *systematic mapping study, survey, and case study* [16]. As software development is rather human-intensive, empirical studies have become common and important in Software Engineering [16][42] as they: (1) integrate human behavior into the empirical methods

employed; (2) provide a scientific basis in software engineering as they help to scientifically evaluate and explain, for example, why one thing is better than another; (3) help to gain knowledge and facilitate knowledge sharing in software engineering. We describe briefly the research methods employed in this thesis.

- Systematic mapping studies focus on providing a wide overview of a domain, identifying research evidence on a topic, and presenting mainly quantitative results [17].
- Surveys aim at collecting information to describe, compare, or explain knowledge, attitudes, and behavior [18].
- Case studies draw on multiple sources of evidence to investigate instances of a contemporary software engineering phenomenon within its real-life context [19].

Table 1 shows a mapping of empirical research methods employed in this thesis to the related knowledge questions and design problems and provides the rationale behind choosing each method.

Table 1. Mapping of empirical methods to the related knowledge questions

Knowledge question	**Empirical method**	**Rationale**	**Chapter**
KQ1. What is the state of the art on assumptions and their management?	Systematic mapping study	The methods of systematic literature review and systematic mapping study are typically used to conduct secondary studies [17]. A systematic literature review aims at identifying, evaluating, and interpreting all available research regarding a certain research question, topic area, or phenomenon of interest [17]. A systematic mapping study focuses on providing a wide overview of a domain, identifying research evidence on a topic, and presenting mainly quantitative results [17]. One of the main differences between a systematic literature review and systematic mapping study is that systematic literature reviews focus on precise research questions, while systematic mapping studies have a broader scope [17]. Assumptions and their management in software development is a broad topic, which covers many different aspects (e.g., software development activities, artifacts, and stakeholders). Thus, we decided to conduct a systematic mapping study instead of a systematic literature review.	2
KQ2. What is the state of the practice regarding architectural assumptions and their management from architects' perception?	Case study (Exploratory)	KQ2 aims to explore a phenomenon, namely architectural assumptions and their management in a real context. Since case studies provide researchers an understanding regarding what actually happens in the real world [19], we decided to conduct an exploratory case study. Furthermore, instead of only getting an overview of architectural assumptions and their management, this knowledge question requires an in-depth analysis, such as coming up with the characteristics of architectural assumptions, which could not be achieved by, for example, a survey [19].	3
DP1.b Is the proposed architectural assumption management process valid?	Case Study (Explanatory)	As mentioned by Runeson *et al.* [19], "*Case studies may be used for explanatory purposes. This involves testing of existing theories in confirmatory studies.*" In this knowledge question, the aim is to "*test an existing theory*", namely to evaluate the proposed architectural assumption management process in software development. Therefore, we decided to conduct an explanatory case study.	4
KQ3. What is the state of the practice regarding Architectural Assumption Description from practitioners' perception?	Survey	KQ3 aims to identify the characteristics from a broad population regarding, for example, how they document architectural assumptions and what challenges they have encountered. We did not aim at exploring what happens when practitioners manage architectural assumptions (in which situation a case study could be employed [19]), or studying correlation or causality of variables related to architectural assumptions (in which situation an experiment is more appropriate [16]). Therefore, we decided to conduct a survey.	5

DP2.c Is the proposed approach valid?	Case Study (Explanatory)	Similarly to DP1.b, the aim of DP2.c is to evaluate the proposed approach for Architectural Assumption Description. Therefore, we decided to conduct an explanatory case study.	6
DP3.b Is the proposed tool valid?	Case Study (Explanatory)	Similarly to DP1.b, the aim of DP3.b is to evaluate the proposed tool for Architectural Assumption Description. Therefore, we decided to conduct an explanatory case study.	7
KQ4. What is the state of the art on the architecture-agility combination?	Systematic mapping study	Similarly to KQ1, the aim of KQ4 is to analyze data from literature, i.e., a secondary study regarding the topic of employing agility into architectural assumption management. Considering the same reasons mentioned in KQ1, we decided to conduct a systematic mapping study.	8
DP4.b Is the proposed approach valid?	Case Study (Explanatory)	Similarly to DP1.b, the aim of DP4.b is to evaluate the proposed approach for architectural assumption management. Therefore, we decided to conduct an explanatory case study.	9

1.6 Overview of the thesis

The rest of the chapters are organized as follows: Chapter 2 to Chapter 9 details the design problems and knowledge questions presented in Section 1.5.1, based on published papers (in peer-review journals or conferences), or papers that are currently under review. Chapter 10 concludes the thesis with future directions. Each chapter is elaborated in the following paragraphs. For Chapter 3, Chapter 6, and Chapter 7, which have been co-authored by other researchers in addition to the supervisors of this thesis, we explain the PhD student's role in the respective paragraphs.

Chapter 2 addresses KQ1, which is based on "*C. Yang, P. Liang, and P. Avgeriou. Assumptions and their management in software development: A systematic mapping study. Information and Software Technology, 94(2): 82-110, 2018.*" The chapter aims to explore and analyze the state of the art on assumptions and their management in software development. It describes a systematic mapping study that covers the literature from January 2001 to December 2015 on assumptions and their management in software development.

Chapter 3 addresses KQ2, which is based on "*C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Heldal, and P. Pelliccione. Architectural Assumptions and their Management in Industry – An Exploratory Study. In: Proceedings of the 11th European Conference on Software Architecture (ECSA), Canterbury, UK, pp. 191-207, 2017.*" The chapter presents an exploratory case study with twenty-four architects to analyze architectural assumptions and their management in industry. In this work, I took the lead in designing and conducting the case study, performing data extraction and analysis, and writing the manuscript.

Chapter 4 addresses DP1, which is based on "*C. Yang, P. Liang, and P. Avgeriou. Evaluation of a process for architectural assumption management in software development. Under review.*" The chapter includes the design of an architectural assumption management process, comprised of four activities, i.e., Making, Description, Evaluation, and Maintenance. It also describes an evaluation of the proposed process: an explanatory case study with 88 first-year master students on software engineering.

Chapter 5 addresses KQ3, which is based on "*C. Yang, P. Liang, and P. Avgeriou. A survey on software architectural assumptions. Journal of Systems and Software, 113(3): 362–380, 2016.*" The chapter describes the current situation on how practitioners document architectural assumptions in software development through a web-based survey with 112 practitioners., who use Chinese as native language and are engaged in software development in China.

Chapter 6 addresses DP2, which is based on "*C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Heldal, P. Pelliccione, and T. Bi. An industrial case study on an Architectural*

Assumption Documentation Framework. Journal of Systems and Software, 134(12): 190-210, 2017." The chapter describes an Architectural Assumption Documentation Framework, which is composed of four viewpoints (i.e., the Detail, Relationship, Tracing, and Evolution viewpoint), to document architectural assumptions in projects. The chapter also includes an evaluation of the framework: a case study with two cases conducted at two companies from different domains and countries. In this work, I took the lead in designing the proposed framework and the case study, conducting the case study in industry, performing data extraction and analysis, and writing the manuscript.

Chapter 7 addresses DP3, which is based on "*C. Yang, P. Liang, P. Avgeriou, T. Liu, and Z. Xiong. Industrial evaluation of an Architectural Assumption Documentation tool: A case study. Under review.*" The chapter introduces a dedicated tool for Architectural Assumption Description. The chapter also includes an evaluation of the tool: an explanatory case study regarding the perceived ease of use and usefulness of the tool with sixteen architects from ten companies in China. In this work, I designed and was involved in the development of the tool, and I took the lead in designing and conducting the case study, performing data extraction and analysis, and writing the manuscript.

Chapter 8 addresses KQ4, which is based on "*C. Yang, P. Liang, and P. Avgeriou. A systematic mapping study on the combination of software architecture and agile development. Journal of Systems and Software, 111(1): 157- 184, 2016.*" The chapter aims to analyze the combination of architecture and agile methods through a systematic mapping study, covering the literature published between February 2001 and January 2014.

Chapter 9 addresses DP4, which is based on "*C. Yang and P. Liang. Identifying and recording software architectural assumptions in agile development. In: Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE). Vancouver, Canada, pp. 308-313, 2014.*" The chapter describes the design of an approach that integrates agility into architectural assumption management. It also includes an evaluation of the proposed approach: an explanatory case study with an architect in China.

Chapter 10 concludes this thesis by summarizing the contributions of the thesis, answers to each knowledge question and design problem, and future directions.