The Journal of Systems and Software 134 (2017) 190–210

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

An industrial case study on an architectural assumption documentation framework

Chen Yang[a,b], Peng Liang[a,*], Paris Avgeriou[b], Ulf Eliasson[c,d], Rogardt Heldal[d], Patrizio Pelliccione[d], Tingting Bi[a]

[a] State Key Lab of Software Engineering, School of Computer Science, Wuhan University, 430072 Wuhan, China
[b] Department of Mathematics and Computing Science, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands
[c] Volvo Cars, Volvo Jacobs Väg, 405 31, Sweden
[d] Department of Computer Science and Engineering, Chalmers University of Technology | University of Gothenburg, 412 96, Sweden

ARTICLE INFO

Article history:
Received 19 July 2016
Revised 1 August 2017
Accepted 2 September 2017
Available online 8 September 2017

Keywords:
Software architecture
Architectural assumption
Documentation framework
Case study

ABSTRACT

As an important type of architectural knowledge, documenting architectural assumptions (AAs) is critical to the success of projects. In this work, we proposed and validated an Architectural Assumption Documentation Framework (AADF), which is composed of four viewpoints (i.e., the Detail, Relationship, Tracing, and Evolution viewpoint), to document AAs in projects. One case study with two cases was conducted at two companies from different domains and countries. The main findings are: (1) AADF can be understood by architects in a short time (i.e., a half day workshop); (2) the AA Evolution view requires the least time to create, followed by the AA Detail view and the AA Relationship view; (3) AADF can help stakeholders to identify risks and understand AAs documented by other stakeholders; and (4) understanding and applying AADF is related to various factors, including factors regarding the framework per se (e.g., tutorial, examples, concepts, and terms), personal experience, resources (e.g., time), tool support, and project context (e.g., project size and number of AAs). Adjusting these factors in an appropriate way can facilitate the usage of AADF and further benefit the projects.

1. Introduction

The concept of assumption is not new in software engineering; several types of assumptions (e.g., requirement assumptions (Haley et al., 2006), architectural assumptions (Roeller et al., 2006), and code-level assumptions (Lehman and Ramil, 2001)) have been discussed in literature. In this work, we focus on architectural assumptions (AAs).

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 AA as architectural knowledge taken for granted, or accepted as true without evidence. The essence of this definition is that it entails *uncertainty* of architectural knowledge: stakeholders are not completely sure about, for instance, the correctness, impact, importance, suitability, or other aspects of architectural knowledge. For example, an architect may assume that "*the concurrent users of the system will exceed 1 million*" . In this statement, the architect is not sure about its correctness, which makes it an AA. This AA will exist until its uncertainty is eliminated.

AAs are not only related to design decisions, but also intertwined with other types of artifacts, such as requirements, use cases, components, design models, design patterns, and risks (Yang et al., 2016, 2017); the lifecycle of AAs spans the whole software development lifecycle, instead of only the architecting or design phase (Yang et al., 2016, 2017).

In our recent industrial survey on AAs (Yang et al., 2016), we investigated the current situation on (1) how practitioners understand both the term and concept of AA and its importance, and (2) whether and how practitioners identify and document AAs in software development. The survey results show that AAs are important in both software architecting and development, and the lack of specific approaches and tools is the major challenge (reason) of (not) identifying and documenting AAs.

According to the literature (e.g., (Rost et al., 2013; Ding et al., 2014, Jansen et al., 2009; Capilla et al., 2015)), architecture documentation in practice is often below expectations in terms of coverage, quality, and accuracy. AAs usually remain implicit and

* Corresponding author.
E-mail addresses: liangp@whu.edu.cn, pliangeng@gmail.com (P. Liang).

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

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

http://dx.doi.org/10.1016/j.jss.2017.09.007
0164-1212/

undocumented in software development (Roeller et al., 2006; Ostacchini and Wermelinger, 2009; Van Landuyt et al., 2012; Fleming and Leveson, 2015; Lago and van Vliet, 2004). We identified two major reasons for this: (1) in the early phases of software development, there is limited information about requirements and design, and the description of the system is normally in natural language, which is informal and may lead to implicit and undocumented AAs (Van Landuyt et al., 2012; Fleming and Leveson, 2015); (2) AAs often stay in stakeholders' mind and they are not aware of them (Roeller et al., 2006; Lago and van Vliet, 2004). As a result, the project team is usually unaware of aspects like (1) which AAs have been made, (2) their implications on the architecture, (3) their state (e.g., valid or invalid), and (4) the relationships between AAs, and between AAs and other types of software artifacts (e.g., requirements).

We also conducted a systematic mapping study on assumptions and their management in software development (Yang et al., 2017). In the mapping study, we defined nine research questions, in which two of them are regarding the challenges of assumptions management and the consequences when assumptions are not well managed in software development. One of the major challenges identified in the mapping study is the "*difficulty to conduct the assumptions management activities, because of, for example, the required effort, incomplete information, or the lack of approaches, tools with guidelines to support the assumptions management activities.*" We also identified twenty-six types of problems caused by implicit assumptions, invalid assumptions, etc. We describe three examples of such problems.

(1) Architectural misunderstanding: For example, stakeholders may misunderstand an architectural design decision, because they are not aware of the AAs behind this decision;
(2) Architectural inconsistency: Conflicting AAs or AAs that are incompatible with other types of software artifacts (e.g., requirements) may result in architectural inconsistency (e.g., between AAs and requirements). Subsequently this may lead to design violations, architecture erosion, and low architecture quality;
(3) Undetected risks. One essential characteristic of assumptions is uncertainty, which may lead to risks in projects. If AAs are not managed, the risks may remain implicit and cause further problems.

Additionally, the results of the mapping study show that assumptions can not only be based on or made for software artifacts, but assumptions can also be software artifacts per se. For example, in (Roeller et al., 2006), the authors defined assumptions as implicit design decisions as well as the rationale and context behind these decisions. Finally, as evidenced by the systematic mapping study, twelve assumptions management activities were identified in the literature, and Assumptions Documentation is the second most frequently discussed activity in all the assumptions management activities. Therefore, we advocate that documenting AAs as part of, for example, architectural decisions documentation is not enough, and AAs need to be treated as first class entities with a dedicated approach for AAs Documentation; this is also supported by several studies (e.g., (Garlan, 2010; Hesse and Paech, 2013)).

In this paper we propose an Architectural Assumption Documentation Framework (AADF), which is composed of four viewpoints: (1) The AA Relationship viewpoint, which describes relationships between AAs. (2) The AA Tracing viewpoint, which describes relationships between AAs and other types of software artifacts. (3) The AA Evolution viewpoint, which documents how AAs evolve over time. (4) The Architectural Assumption Detail viewpoint, which provides all detailed information about AAs. These four viewpoints frame 23 AA concerns, which were identified and collected from the existing literature, and refined by the first three authors and six architects from industry in a three-round selection process. Furthermore, we conducted an industrial case study with two cases of two companies (i.e., IBO Technology[3] and Volvo Cars[4]) from two domains (Internet of Things and Automotive industry respectively) in two countries (China and Sweden respectively) to validate the effectiveness of AADF.

The results indicate that (1) AADF can be understood in a short time (i.e., a half day workshop); (2) the AA Evolution view requires the least time to create, followed by the AA Detail view and the AA Relationship view; (3) AADF can help stakeholders to identify risks and understand AAs in projects (the AAs documented by other stakeholders); and (4) understanding and applying AADF is related to various factors, including factors regarding the framework per se (e.g., tutorial, examples, concepts, and terms), personal experience, resources (e.g., time), tool support, and project context (e.g., project size and number of AAs). We exemplify two of these factors: (a) experienced software engineers (e.g., architects) can understand and use AADF easier and better than junior software engineers; (b) Tool support is critical for the application of AADF in practice, for example, to deal with different project context and provide good quality of outputs. Adjusting these factors in an appropriate way can facilitate the usage of AADF and further benefit the projects.

The rest of the paper is structured as follows: Section 2 describes related work on software assumptions and their documentation. Section 3 introduces AADF in detail. Section 4 presents the case study design used to evaluate the effectiveness of AADF. Section 5 describes the results of the case study and Section 6 presents the interpretations and implications of the results. Section 7 discusses the threats to the validity of the case study, and Section 8 concludes this work with future directions.

2. Related work

The works presented in this section first investigate different types of assumptions in software development as well as their documentation, and then compare them to our work.

2.1. Architectural assumptions and their documentation

Roeller et al. (2006) proposed an approach for AAs Recovery, and used a template with three elements "Suspicious effect", "Interview", and "Assumption" to document AAs in projects. Garlan et al. (2009) treated AAs as an important factor that causes architectural mismatch. The authors suggested that guidelines should be provided for documenting AAs (e.g., how to integrate AAs Documentation into architecture documentation). The authors further summarized several approaches (e.g., architecture views and description languages) and techniques (e.g., XML) to support AAs Documentation. Van Landuyt et al. (2012) discussed a specific type of AAs (i.e., early AAs), which are made by requirements engineers in the early phases of development (e.g., requirements elicitation). The authors highlighted the necessity for the documentation of early AAs. In their subsequent work, Van Landuyt and Joosen (2014) introduced a metamodel and an instantiation strategy to document early AAs based on quality attribute scenarios and use cases. Ordibehesht (2010) argued that implicit and invalid AAs are the major cause that leads to system failures and poor performance. The author proposed an approach based on an architectural analysis and description language to document AAs in software development. Mamun and Hansson (2011) conducted a literature review on the topic of assumptions in the context of software development. In this review, the authors identi-

[3] http://www.ibotech.com.cn/.
[4] http://www.volvocars.com/.

fied problems (e.g., architectural mismatch), challenges (e.g., distinguishing assumptions from other software artifacts), and approaches (e.g., assumption description language) for assumptions management (e.g., Assumptions Documentation). In their following work, Mamun et al. (2012) proposed to use Alloy language to document AAs in software development. In our previous work (Yang and Liang, 2014), we focused on AAs and their documentation in agile development, by proposing a simplified conceptual model for AAs with a lightweight approach for AAs Documentation. Furthermore, we surveyed 112 practitioners on AAs in software development (Yang et al., 2016). The results of the survey show that most AAs are kept implicit due to the lack of documentation. Furthermore, the lack of specific approaches and tools is the major challenge (reason) of (not) identifying and documenting AAs.

2.2. Other types of assumptions and their documentation

Lewis et al. (2004) identified several problems caused by not documenting assumptions in projects. For example, if assumptions are implicit, stakeholders may not be aware of the evolution of these assumptions (e.g., valid assumptions turning out to be invalid over time). Additionally, the authors developed an Assumption Management System for Assumptions Documentation in software development. Lago and van Vliet (2005) noted that stakeholders make assumptions about what will (or will not) change during development, and these assumptions are connected to system variability and invariability. The authors designed a metamodel for Assumptions Documentation, and showed how to model assumptions in an explicit way based on a product family. Ostacchini and Wermelinger (2009) emphasized that, in agile development, most assumptions are implicit, and explicitly documenting assumptions is important. The authors developed a process for assumptions management (e.g., using an assumption model to document assumptions) in agile development. Furthermore, the authors summarized the lessons learned for managing assumptions (e.g., Assumptions Documentation). Steingruebl and Peterson (2009) introduced several specific approaches and techniques to document different types of assumptions in software development. For example, the authors suggested that stakeholders can use requirements documentation, dataflow diagrams, and threats modeling to document software design assumptions, and use source code annotations and contract languages (e.g., Eiffel) to document implementation assumptions. Haley et al. (2006) concentrated on trust assumptions, and pointed out that the essence of these assumptions is to trust the characteristics of domains. The authors emphasized that trust assumptions can impact the security of systems (i.e., security requirements) and should be documented in software development. The authors also proposed a framework to document trust assumptions with security requirements in projects. Miranskyy et al. (2005) traced assumptions to requirements, and designed a quantitative and mathematical model (using Boolean network and stochastic processes) to document these assumptions with requirements in software development. Tirumala et al. (2005) mentioned that many component assumptions are implicit and undocumented in projects, and not documenting component assumptions may lead to several problems (e.g., component mismatch). The authors proposed a framework with a dedicated tool using XML, to document component assumptions in software development. Uchitel and Yankelevich (2000) described the problems of component assumptions and developed guidelines on how to use the existing architectural description languages to document component assumptions. For example, the authors highlighted that component assumptions documentation should be independent, instead of, for example, combining it into component behavioral description. Lehman and Ramil (2001) discussed the role of assumptions at the implementation level, as well as the problems caused by these assumptions. For example, the authors stated that invalid assumptions could lead to uncertain and/or unacceptable program behavior, and documenting these assumptions is a solution to address such problems. Furthermore, the authors noted several lessons learned for assumptions management (e.g., assumptions should be documented iteratively in various phases of the development). Zschaler and Rashid (2011) focused on aspect assumptions in aspect-oriented software development (e.g., assuming the context in which the aspects are used). In their work, the authors used a template with four elements “Name”, “Description”, “Examples”, and “Discussion” to document aspect assumptions.

2.3. Comparison to related work

The majority of the related work does not deal with AAs but with other types of assumptions within different phases of software development. Assumptions Documentation approaches, techniques, and tools designed for other phases of software development may not be suitable for AAs Documentation. For example, our systematic mapping study (Yang et al., 2017) reports that in the context of software design, architects and designers are the major stakeholders in assumptions management. Therefore, we considered that managing AAs in a project is mainly the responsibility of architects, designers, and architecture reviewers; for other types of assumptions, the stakeholders could be diverse (e.g., requirements engineers, developers, and project managers).

There is also related work that targets AAs and their documentation specifically. We see the following limitations in the work: (1) Different stakeholders have various AA concerns, but the existing approaches, techniques, and tools only address few of them, and the connections between AA concerns and stakeholders are not clear; and (2) It is unclear which AA concerns are addressed by the proposed approaches, techniques, and tools, and how they address the concerns.

Additionally, AADF in this work can be compared to other Architecture Frameworks, such as the Architecture Decision Documentation Framework proposed by van Heesch et al. (2012). The only common element between these two frameworks is that both of them follow the same standard – ISO/IEC/IEEE Std 42010–2011 (ISO, 2011); all the other parts of the design of the two frameworks are different. First, the two frameworks were developed based on different motivations. The motivation of this work is based on our industrial survey on AAs (Yang et al., 2016) and a systematic mapping study on assumptions and their management in software development (Yang et al., 2017). Furthermore, the two frameworks used different process for identifying and selecting concerns. We collected 78 concerns on assumptions from the existing literature. 23 AA concerns were finally selected and four categories of the concerns (i.e., addressed by the four AADF viewpoints) were classified by using Constant Comparison (Glaser and Strauss, 1967). Finally, the metamodels in the two frameworks are completely different.

AADF is the first approach that proposes a framework for AAs Documentation. This framework aims at documenting AAs in an explicit and systematic way based on the AA concerns of stakeholders (see Section 3.1). AADF not only provides a template to document the details of each AA, but also supports stakeholders to get an overview of the relationships between AAs, trace AAs to other types of software artifacts, and monitor the evolution of AAs in software development.

3. A framework for architectural assumption documentation

In this work, we advocate treating AAs as first class entities in architecture documentation, and we focus on documenting them – we do not focus on documenting other types of artifacts such as

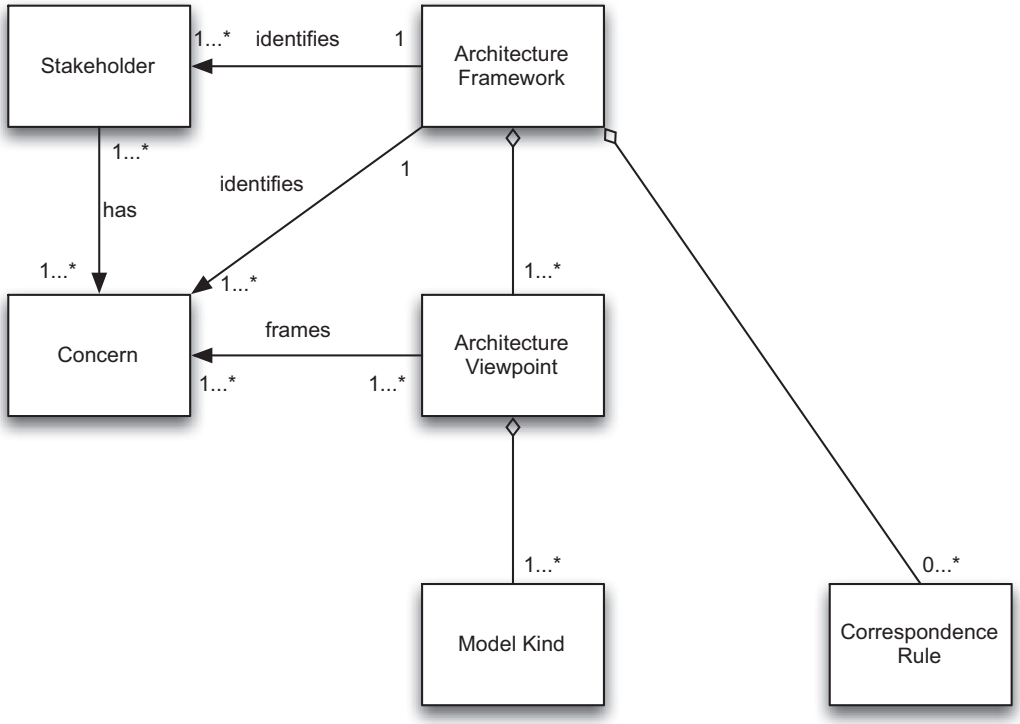

Fig. 1. Conceptual model of architecture framework from ISO/IEC/IEEE 42010 (ISO, 2011).

decisions. An architecture framework "*establishes a common practice for creating, interpreting, analyzing, and using architecture descriptions within a particular domain of application or stakeholder community.*" (ISO, 2011). As shown in Fig. 1, an architecture framework is composed of a set of viewpoints, which can be used to address specific concerns of stakeholders. In this work, we developed AADF based on the guidelines proposed in ISO/IEC/IEEE 42010 (ISO, 2011) using the following process (further elaborated in Section 3.1): (1) we identified 24 AA concerns in a three-round selection process with six architects. We established that one concern cannot be framed in a single viewpoint and was therefore removed, namely "*Which AA should be maintained during the software engineering lifecycle? (AA maintenance means the modification of assumptions, e.g., fix the conflicts between assumptions and other software artifacts, modify inconsistent assumptions, and remove invalid assumptions)*" ; (2) we identified eight types of stakeholders with two architects; (3) we related the eight types of stakeholders to the 23 concerns with two architects; and (4) we developed four viewpoints that frame the 23 concerns by using Constant Comparison (following the guidelines proposed by Glaser and Strauss (Glaser and Strauss, 1967)). As mentioned in literature (e.g., (Glaser and Strauss, 1967; Adolph et al., 2011)), Constant Comparison provides a systematic way to generate, for example, concepts from incidents, and a continuous process of verification of the generated concepts and categories. We coded the concerns as incidents, compared these incidents to each other to generate concepts, and further performed comparison among incidents and concepts to generate categories. This process was iteratively conducted by the first author. The second and third author reviewed the results of each iteration.

Section 3.1 presents the concerns for architectural assumptions. The elements as well as the concerns addressed by the corresponding four viewpoints are elaborated in Sections 3.2 to Section 3.5 respectively. Table 16 in Appendix B further shows definitions of context elements (e.g., requirement, artifact, and risk) related to AADF. Section 3.6 provides guidelines on using AADF, including the correspondence rules, the essential, important, and optional elements of AADF, and the benefits and costs of using AADF.

3.1. Concerns for architectural assumptions

AAs should be documented to support software development (Yang et al., 2016). However, there is a lack of widely accepted understanding on the concerns of stakeholders that should be addressed when documenting AAs. A concern is any interest in a system related to stakeholders (ISO, 2011).

We identified and collected 78 related concerns from the existing literature, which is part of our earlier work (see the systematic mapping study (Yang et al., 2017)). We further conducted a three-round selection to refine the concerns as detailed below.

(1) In the first round, the first three authors merged the concerns that have a similar meaning; 44 concerns were left in this round. Note that the activity was iterated also in the other rounds of selection.
(2) In the second round, the second and third author ranked the importance of the 44 concerns according to their experience and knowledge on AAs in software development through a questionnaire using a 5-point Likert-scale (William and Donnelly, 2006) (i.e., 1 being the least important and 5 being the most important). If a concern got a ranking below 3 by both the second and third author, the first three authors further discussed whether this concern should be removed. 7 concerns were removed in this round.
(3) In the third round, we invited six architects from industry to rank the importance of the 37 concerns according to their experience and knowledge on AAs in software development also through a questionnaire using a 5-point Likert-scale (i.e., 1 being the least important and 5 being the most important). All six

Table 1
Concerns for architectural assumptions.

ID	Concern
C1	Which AAs have been made?
C2	What risks are caused by assumption A?
C3	Which stakeholders are affected by assumption A?
C4	Which AAs are influenced by stakeholder S? (E.g., we may need to revisit some assumptions, when stakeholder S no longer has a stake in the project or changes his/her stake.)
C5	Which AAs conflict with assumption A?
C6	Which AAs are constrained by assumption A?
C7	Which AAs are caused by assumption A?
C8	Which AAs are the alternatives of assumption A?
C9	Which AAs are strengthened by assumption A?
C10	Which AAs are weakened by assumption A?
C11	Which AAs are related to assumption A?
C12	What is the relationship between Assumption A and Requirement R?
C13	What is the relationship between Assumption A and Architectural Design Decision D?
C14	What is the relationship between Assumption A and Component C?
C15	What is the relationship between AAs and changes in the system? (E.g., changes on requirements, architectural design decisions, or components)
C16	Which AAs change and when? (AAs can evolve over time, for example, turning from valid to invalid during its lifecycle.)
C17	What is the plan when an assumption A changes? (E.g., when assumption A turns to be invalid, stakeholders may need to make a plan, instead of modifying it immediately.)
C18	Which AAs are violated (conflict with other artifacts) during the software engineering lifecycle?
C19	How did assumption A change over time?
C20	What is the state (e.g., "Modified" and "Valid") of assumption A?
C21	What is the rationale behind assumption A?
C22	What are the pros and cons of assumption A?
C23	What makes assumption A invalid (the reasons)?

architects had at least six years of working experience in IT industry, ranging from 6 to 34 years (average: 15.67 years); and at least four years of working experience in software architecting, ranging from 4 to 22 years (average: 10.33 years). If a concern got a ranking below 18 in total, the first three authors further discussed to decide whether this concern should be removed.

Table 1 shows the selected AA concerns. We acknowledge that certain steps in the selection process of AA concerns are subjective and may have introduced bias. This may pose risks to the design of AADF. To make this process as transparent as possible, we made the materials of this work available online (Yang et al., 2017).

As mentioned in ISO/IEC/IEEE Std 42010–2011 (ISO, 2011): "*A concern could be held by one or more stakeholders. In general, the association of concerns with stakeholders is many-to-many.*" The AA concerns and their stakeholders act as a basis for the development of AADF. We applied the following process to identify stakeholders and assign stakeholders to the AA concerns:

(1) We interviewed the two architects who participated in the third-round selection of AA concerns, and asked them about the potential stakeholders who may have an interest in AAs and their documentation. Eight types of stakeholders were identified.
(2) The first author and the two architects assigned stakeholders to concerns after stakeholders identification, and the other authors reviewed the results (see Table 2).

Table 2
Stakeholders and their concerns for architectural assumptions.

Stakeholder	Concerns for architectural assumptions
Project manager	C1, C2, C17, C20, C22
Requirements engineer	C1, C12, C15, C17, C20
Architect & Designer	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23
Developer	C1, C3, C14, C15, C17, C20
Tester	C1, C14, C15, C17, C20
Maintainer	C1, C3, C14, C15, C17, C20
Customer & User	C1, C2, C12, C20
Architecture reviewer	C1, C2, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23

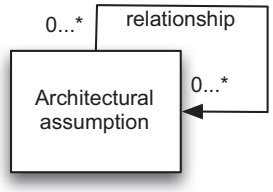

Fig. 2. Metamodel of Architectural Assumption Relationship viewpoint.

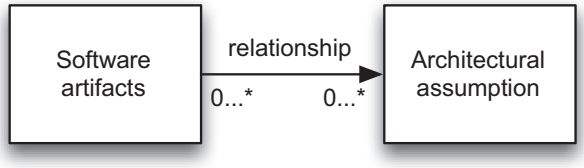

Fig. 3. Metamodel of Architectural Assumption Tracing viewpoint.

3.2. Architectural assumption relationship viewpoint

The AA Relationship viewpoint is used to capture the relationships between AAs (in Table 3). The metamodel of this viewpoint is shown in Fig. 2. The relationship types in the AA Relationship viewpoint are based on the identified AA concerns.

3.3. Architectural assumption tracing viewpoint

AAs are related to various types of software artifacts such as requirements and architectural design decisions. The AA Tracing viewpoint presents such relationships (in Table 4). Fig. 3 shows the metamodel. To make the relationship types between AAs and between AAs and other types of software artifacts consistent, we

Table 3
Relationships between architectural assumptions.

Relationship	Description	Concerns
A *conflicts with* B	A symmetrical relationship that indicates that the two assumptions A and B are mutually exclusive.	C5
A *constrains* B	Assumption B is tied to Assumption A. If Assumption A is changed, then Assumption B should be revisited.	C6
A *is caused by* B	Assumption B causes the making of Assumption A.	C7
A *is an alternative to* B	Assumption A and B can be a substitute of each other.	C8
A *strengthens* B	Assumption A increases the possibility of Assumption B being valid.	C9
A *weakens* B	Assumption A decreases the possibility of Assumption B being valid.	C10
A *is related to* B	There is a relation between the two AAs, and the type of this relationship is not covered by the types listed above.	C11

Table 4
Relationships between architectural assumptions and other software artifacts.

Relationship	Description	Concerns
A *conflicts with* B	A symmetrical relationship indicating that artifact A and assumption B are mutually exclusive.	C2, C12, C13, C14
A *constrains* B	Assumption B is tied to Artifact A. If Artifact A is changed, then Assumption B should be revisited or vice versa.	
A *is caused by* B	Artifact A causes the making of Assumption B or vice versa.	
A *strengthens* B	Artifact A increases the possibility of assumption B being valid.	
A *weakens* B	Artifact A decreases the possibility of assumption B being valid.	
A *is related to* B	There is a relation between artifact A and assumption B, and the type of this relationship is not covered by the types listed above.	

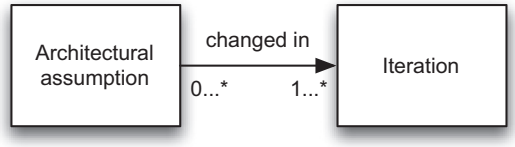

Fig. 4. Metamodel of Architectural Assumption Evolution viewpoint.

used the same set of relationships in the AA Tracing viewpoint, except for the relationship type "*is an alternative to*", because an AA cannot be an alternative to another type of artifacts.

The AA Tracing viewpoint traces AAs to four types of software artifacts: (1) requirements (C12); (2) architectural design decisions (C13); (3) system components (C14); and (4) risks (C2).

3.4. Architectural assumption evolution viewpoint

Similar to other types of assumptions in software engineering, AAs have a dynamic characteristic: the system as well as the project context (e.g., market), is changing over time, possibly making assumptions invalid, which may cause problems in projects (Lewis et al., 2004). For example, consider a stakeholder assuming that a third-party component can be used in her/his project. However, during development, the stakeholder discovers that some problems caused by the component cannot be addressed, because of the lack of key information (e.g., design rationale) of the component. In this case, the original AA: "*the third-party component can be used in this project*" turns out to be invalid. Furthermore, AAs may be invalid in the first place (at the time they are made), because of, for example, the lack of knowledge or information when making these AAs. Therefore, the AA Evolution viewpoint aims to document how AAs evolve over time. Fig. 4 shows the metamodel of the AA Evolution viewpoint, which is composed of two elements (see Table 5). An example about how to use the AA Evolution viewpoint is provided in Table 14 of Appendix A and Fig. 6.

3.5. Architectural assumption detail viewpoint

Details of an AA can be valuable to stakeholders but also take more effort to document – it contains more information than the other viewpoints thus it also requires comparatively more effort. The AA Detail viewpoint aggregates all information of an AA from the rest of the viewpoints and is complemented with other optional details that may be documented if resources are available. The aim was to make the AA Detail viewpoint provide an overview of each AA, while the other three viewpoints have their specific

Table 5
Elements of Architectural Assumption Evolution viewpoint.

Element	Description	Concerns
Architectural assumption	ID: The ID of an AA (e.g., AA1), requirement (e.g., R1), architectural design decision (e.g., ADD1), component (e.g., Cp1), or risk (e.g., R1) State includes two groups: Validation state = {"Valid" and "Invalid"}, and Action state = {"Added", "Modified", "Transformed", and "Removed"}. Validation state is used to show whether an AA is valid, and Action state is used to show whether an AA has been added, modified, transformed, or removed.	C15, C16, C18, C19, C20
Iteration	Iteration is the act of repeating a process with the aim of approaching a desired goal, target or result. It can be adapted to other variables such as "Version".	C16, C19

focus: the AA Relationship viewpoint shows the relationships between AAs, the AA Tracing viewpoint traces AAs to other types of software artifacts, and the AA Evolution viewpoint provides the history of an AA during its lifecycle. Despite this redundancy of information across viewpoints, the objective is to support engineers in documenting AAs without entering redundant information (e.g., once the historical information is recorded in the AA Evolution view, the AA Detail view can be updated automatically). Effective tool support is vital in this case, and is considered as our next step (see Section 8).

The AA Detail viewpoint is composed of eight elements (see Table 6). Fig. 5 shows the metamodel of the AA Detail viewpoint, and Table 15 in Appendix A shows a real example of AA Detail viewpoint from industry. To make the example simple and easy to understand, "Related architectural assumptions", "Related requirements", "Related architectural design decisions", "Related system components", "Related risks", and "History" are not included.

3.6. Guidelines on using AADF

This section first presents the correspondence rules of AADF, then classifies the AADF elements as essential, important or optional, and finally provides a set of guidelines regarding benefits and costs of using AADF based on our own experience and knowledge.

3.6.1. Correspondence rules of AADF

As mentioned in ISO/IEC/IEEE 42010 (ISO, 2011): "*Correspondence rules are used to enforce relations within an architecture description (or between architecture descriptions).*" Table 7 shows the fundamental correspondence rules of AADF and respective viewpoints.

3.6.2. Essential, important, and optional elements of AADF

As mentioned in Section 3.1, the second and third author and six architects were involved in ranking the importance of the AA concerns using Likert-scale scores. The first author further classified the 23 AA concerns in three groups according to their scores.

(1) Group 1 includes the AA concerns that received a score of less than 32 (i.e., avg. < 4): C3, C7, C8, C11, C17, and C19.
(2) Group 2 includes the AA concerns that received a score between 32 and 36 (not including 36, i.e., 4 <= avg. < 4.5): C1, C4, C5, C9, C10, C12, C16, C18, C20, C22, and C23.
(3) Group 3 includes the AA concerns that received a score of at least 36 (i.e., avg. >= 4.5): C2, C6, C13, C14, C15, and C21.

Table 6
Elements of Architectural Assumption Detail viewpoint.

Element	Description	Concerns
Architectural assumption	The properties of an AA include ID, Name, Description, State, Rationale, Pros, Cons, and Plan. State contains two groups: Validation state = {"Valid" and "Invalid"} and Action state = {"Added", "Modified", "Transformed", and "Removed"}. Validation state: whether the AA is valid. Action state: whether the AA has been added, modified, transformed, or removed. Rationale: The reasons for making the AA. Pros: The positive impacts of the AA. Cons: The negative impacts of the AA. Invalidity reason: The reasons about why the AA is invalid. Plan: The plan for the AA, including "To be modified", "To be removed", and "To be transformed (to another type of artifact)".	C1, C17, C20, C21, C22, C23
Stakeholder	The stakeholders who are involved in the AA	C3, C4
Related architectural assumptions	The relationships between the assumption and other assumptions. The information of this element can be collected from the Architecture Assumption Relationship views.	C5, C6, C7, C8, C9, C10, C11
Related requirements	The relationships between the assumption and requirements. The information of this element can be collected from the Architecture Assumption Tracing views.	C12
Related architectural design decisions	The relationships between the assumption and architectural design decisions. The information of this element can be collected from the Architecture Assumption Tracing views.	C13
Related system components	The relationships between the assumption and system components. The information of this element can be collected from the Architecture Assumption Tracing views.	C14
Related risks	The relationships between the assumption and risks. The information of this element can be collected from the Architecture Assumption Tracing views.	C2
History	The evolution of the assumption. The information of this element can be collected from the Architecture Assumption Evolution views.	C15, C16, C18, C19, C20

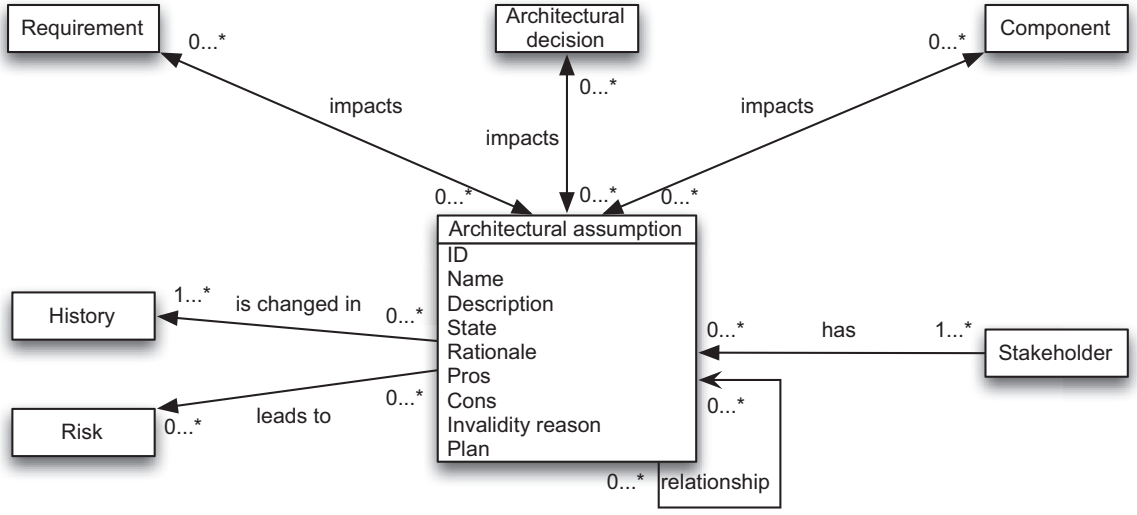

Fig. 5. Metamodel of Architectural Assumption Detail viewpoint.

Table 7
Fundamental correspondence rules of AADF.

Correspondence rule	Viewpoints
The relationships between two AAs or an AA and another type of artifact (i.e., requirement, design decision, component, or risk) can be zero, one, or multiple.	Relationship, Tracing, Detail
The relationships between two AAs or an AA and another type of artifact (i.e., requirement, design decision, component, or risk) can be bi-directional.	Relationship, Tracing, Detail
If an AA is valid, the "Invalid reason" of the AA must be "N/A".	Detail
The field "Related architectural assumptions" of an AA in an AA Detail view should be consistent with the relationships between the AA and other AAs in the AA Relationship views.	Relationship, Detail
The field "Related requirements", "Related architectural design decisions", "Related system components", and "Related risks" of an AA in an AA Detail view should be consistent with the relationships between the AA and other types of artifacts (i.e., requirement, design decision, component, or risk) in the AA Tracing views.	Tracing, Detail
The field "History" of an AA in an AA Detail view should be consistent with the evolution of the AA across time in the AA Evolution views.	Evolution, Detail
The latest version of an AA in the Relationship, Tracing, and Detail view should correspond to the latest occurrence of the AA in the Evolution view.	All
The ID of a certain AA in different views should be consistent.	All
An AA can be documented in zero, one, or multiple Relationship, Tracing, Evolution, or Detail views.	All

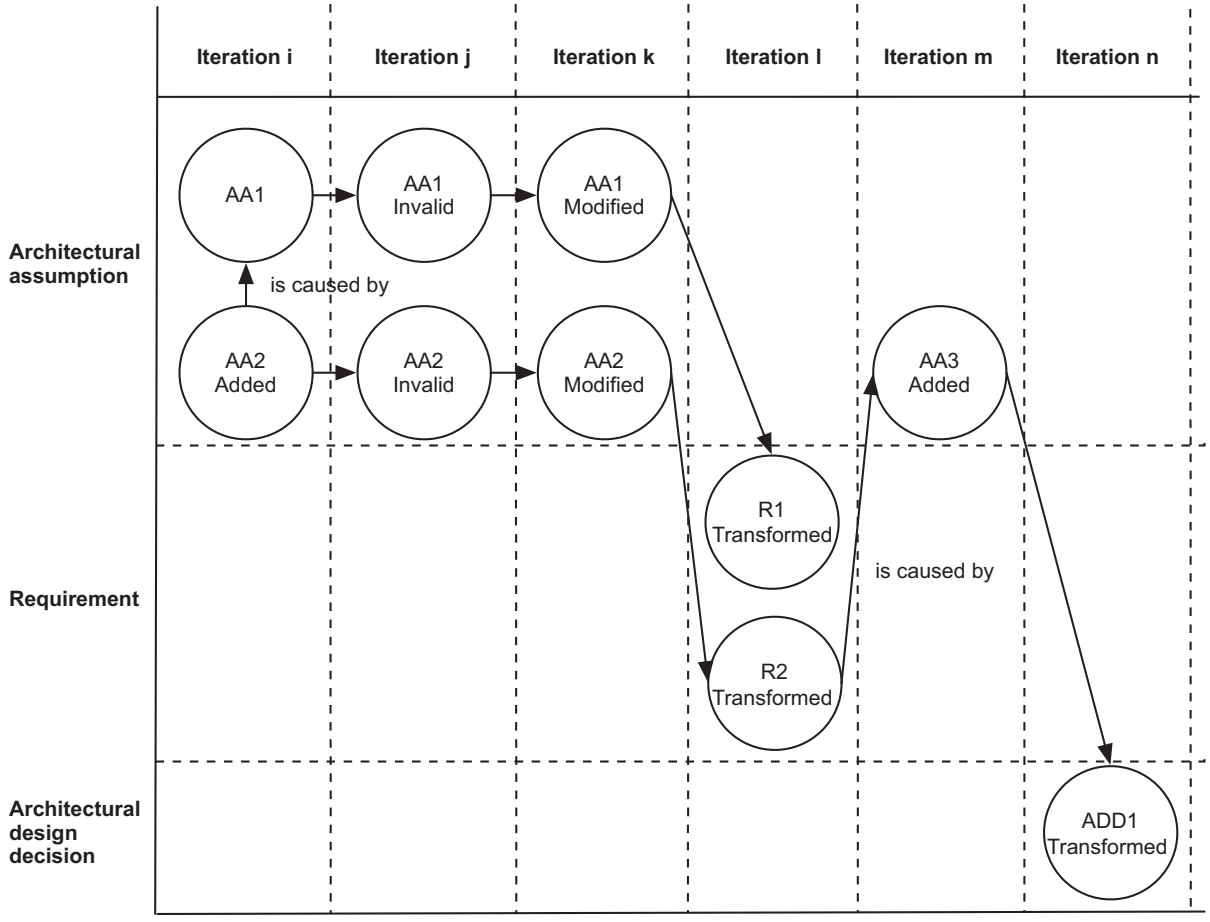

Fig. 6. An example of the Architectural Assumption Evolution viewpoint.

Table 8
Importance of the elements in AADF.

Element	Viewpoint	Importance	Concerns
ID of an artifact (e.g., AA)	All	Essential	All
AAs with a "constrains" relationship	Relationship, Detail	Essential	C6
Relationships between AAs and architectural design decisions	Tracing, Detail	Essential	C13
Relationships between AAs and components	Tracing, Detail	Essential	C14
Relationships between AAs and risks	Tracing, Detail	Essential	C2
State of an AA	Evolution, Detail	Essential	C15, C18, C19, C20
Rationale of an AA	Detail	Essential	C21
Name of an AA	Detail	Important	C1
Description of an AA	Detail	Important	C1
Pros of an AA	Detail	Important	C22
Cons of an AA	Detail	Important	C22
Invalidity reason of an AA	Detail	Important	C23
Stakeholder	Detail	Important	C3, C4
AAs with a "conflicts with" relationship	Relationship, Detail	Important	C5
AAs with a "strengthens" relationship	Relationship, Detail	Important	C9
AAs with a "weakens" relationship	Relationship, Detail	Important	C10
Relationships between AAs and requirements	Tracing, Detail	Important	C12
Iteration	Evolution, Detail	Important	C16, C19
AAs with a "is caused by" relationship	Relationship, Detail	Optional	C7
AAs with a "is an alternative to" relationship	Relationship, Detail	Optional	C8
AAs with a "is related to" relationship	Relationship, Detail	Optional	C11
Plan of an AA	Detail	Optional	C17

Through mapping the AA concerns in these three groups to the elements in AADF, we concluded the importance of the elements in AADF as shown in Table 8. Note that an AADF element can relate to multiple concerns (e.g., "Stakeholder" in the AA Detail viewpoint can be mapped to both C3 and C4). We classified the importance of such elements according to the most important concern the elements being mapped to (e.g., we classified the importance of "Stakeholder" as "Important" instead of "Optional" because C4 is in Group 2).

3.6.3. Benefits and costs of using AADF

AADF has both benefits and costs. For example, AADF can help stakeholders to get an overview of the AAs made in projects, but the effort required could be prohibitive. There is always a trade-

Table 9
Relationships among tasks, subjects, inputs, outputs, and RQs.

Part	Task	Who	Inputs	Outputs	RQs
Part 1	T1	Researchers and architects	(1) The supporting documents (e.g., the AADF design) (2) The documents of the questionnaire	(1) The cooperation agreement (2) The case study design (3) The documents of the tutorial (4) The results of the questionnaire	N/A
	T2	Researchers and architects	(1) The case study design (2) The documents of the tutorial	(1) The questions and answers about the case study	N/A
Part 2	T3	Researchers and architects	(1) The documents of the selected projects (e.g., requirements documents) (2) The Excel template of AADF (3) The documents of the tutorial	(1) The AADF views (2) The questions and answers about the case study	RQ1, RQ2, RQ3
	T4	Researchers and architects	(1) The AADF views	(1) The questions and answers about the case study	RQ4
Part 3	T5	Researchers and architects	(1) The documents of the interview and focus group	(1) The results of the interview (2) The results of the focus group	RQ1, RQ2, RQ3, RQ4

off between the benefits and costs of using AADF in projects. Below we provide guidelines as derived from our own experience and knowledge in using AADF:

(1) AAs are intertwined with various types of software artifacts, such as requirements, design decisions, and components, and their lifecycle spans the whole software development. We advocate treating AAs as first class entities in architecture documentation when using AADF.
(2) The AA concept is subjective, thus stakeholders may understand it differently. Before using AADF, stakeholders in the same project or organization should at least reach an agreement on the understanding of the elements in AADF, especially the AA term and concept.
(3) AAs documentation does not depend on existing documentation of other types of artifacts.
(4) AADF can potentially be extended by additional viewpoints in order to frame other concerns. Furthermore, AADF is not a unique solution in framing the stated concerns. This is in accordance to ISO/IEC/IEEE 42010 (ISO, 2011), which states that there is no unique set of viewpoints for addressing a specific set of concerns: one could consider other viewpoints that frame the same concerns.
(5) AADF is completely flexible and customizable: users of AADF do not need to use all the viewpoints and elements in AADF. Instead they can choose the elements in certain viewpoints to be documented. For example, they can document only key information according to their project context in order to save time and effort.
(6) AAs management can be resource-intensive; AADF may need to be customized, in order to be used in more agile types of development.
(7) AADF may not be useful in projects that do not have many AAs to manage, such as small projects or "stable" projects (e.g., a project that is similar to a previous project with few uncertainties).
(8) AADF is particularly suitable in developing critical systems (e.g., safety-critical systems). Stakeholders need to pay significant attention to AAs management and use systematic approaches to manage AAs in this type of systems, since implicit or invalid AAs may cause serious issues.
(9) Not every AA needs to be documented with AADF. Stakeholders should identify the important AAs in their projects before using AADF.
(10) AADF needs to be used from the early phases of software development (e.g., requirements engineering or architecture analysis); the usage of AADF spans the whole development lifecycle.
(11) Though the major focus of AADF is AAs Documentation, AADF can also be used to support other AAs management activities. For example, in AAs Evaluation, stakeholders can use the documentation managed by AADF to evaluate each documented AA.

4. Case study

We followed the guidelines proposed by Runeson and Höst (2009) to design and report on this case study. Furthermore, we made the materials of this work available online (Yang et al., 2017).

4.1. Goal and research questions

The goal of the case study is to **analyze** AADF **for the purpose** of evaluation **with respect to** its effectiveness in AAs Documentation **from the point of view of** architects **in the context of** software development in industry. The effectiveness of AADF for AAs Documentation can be evaluated in four aspects as detailed below:

(1) **Ease of understanding the AADF viewpoints**. AADF is composed of four viewpoints. The ease of understanding the AADF viewpoints has a significant impact on the adoption of AADF in AAs Documentation. Without a fair understanding of the AADF viewpoints, stakeholders may use AADF improperly and inefficiently in their projects.
(2) **Effort of documenting AAs using the AADF viewpoints**. Creating the AADF views takes some effort. It will not be acceptable for stakeholders if they have to spend too much effort on documenting AAs.
(3) **Effectiveness in helping stakeholders to identify risks in projects**. Identifying risks is critical to the success of projects (Boehm, 1991). The most important characteristic of AAs is "uncertainty", and AAs may lead to various risks in software development. Therefore, being aware of the risks related to AAs is important in development.
(4) **Effectiveness in helping architects to understand AAs in projects**. Understanding AAs in projects means being aware of the AAs made in software development, their state, rationale, pros, cons, invalidity reasons, the related stakeholders, plans, the relationships between AAs, the relationships between AAs and requirements, design decisions, components, and risks, and the evolution of each AA.

Aspects (1) and (2) concern *producing AAs* (i.e., creating the AADF views), while aspects (3) and (4) are about *consuming AAs* (i.e., using the created AADF views) (Tang et al., 2010). The research questions (RQs) of this study, which are formulated based on the four aspects of the effectiveness of AADF in AAs Documentation, are described as follows.

RQ1: How easy is it to understand the AADF viewpoints?
RQ2: What is the expected effort for creating the AADF views?

Table 10
AAs documented with AADF by subjects.

P1	P2	P3	P4
9200 ultra-high frequency component	UCOSII operation system	433 MHz component	13.56 MHz passive sensing head
R2000 ultra-high frequency component	Mobile telecommunication	GPS component	Direction of 5.8GHz
Voice broadcast component	GPS component	GSM component	Security of wireless communication
Internet communication	SDK of EFM32	Power component	Conditions of payment
RS485 communication		Polymer battery	Response time of wireless communication
IAP			Success rate of wireless communication
Wifi component			Personal identification component
LED component			
P5	P6	P7	P8
Power-supply system	Number of parking spots per meter	ASIL and Infotainment	Subsystem structure
Distance of ultrasonic wave sensors	License plate recognition	Updateability of domain masters	Awareness of architectural strategies
433 MHz component	Communication between meters and servers	Functional growth	Evolution of architecture
Time of sending and receiving ultrasonic wave	Communication between vehicle detectors and meters	ASIL and Safety	
		System safety decomposition	

RQ3: Does AADF effectively help architects to identify risks in projects?
RQ4: Does AADF effectively help architects to understand AAs in projects?

We note that RQ2 is concerned with the effort that architects expected to spend on creating the AADF views and not the actual effort spent during the case study. Since the data collection was conducted during a half-day workshop at each of the company instead of the whole lifecycle of a project due to resource constraints, the time for data collection was strictly limited. To ensure that the subjects used every AADF viewpoint in the case study, we let the subjects spend equal time on creating each AADF view. Therefore, we did not measure their effort quantitatively. Instead we asked them about their expected effort of using AADF: which views they would expect to be the most or least time-consuming to create when using AADF in their work.

4.2. Case and subject selection

We have studied a phenomenon (use of the AADF) in a real context, by asking all the subjects to select one non-trivial software project from their work and used AADF in the context of their own projects. Furthermore, as mentioned by Runeson and Höst (2009), "*Case studies may be used for explanatory purposes. This involves testing of existing theories in confirmatory studies.*" In this work, the aim was to "test an existing theory", namely to evaluate the effectiveness of AADF in AAs Documentation. Therefore, it is an explanatory case study.

4.2.1. Case description

A case is a contemporary phenomenon in its real-life context (Yin, 2003). We selected and conducted two cases in this case study. One case was conducted in China with six architects from the domain of Internet of Things at IBO Technology, and another case was conducted in Sweden with six architects from the domain of Automotive Industry at Volvo Cars. Note that the six architects who had been involved in the three-round selection process of AA concerns were not among the subjects of the case study. IBO Technology is a Chinese company headquartered in Shenzhen, China, and focuses on IT services (e.g., system integration services) and Internet of Things. It has 100 – 200 employees (around 80% are engineers). Volvo Cars is a Swedish premium automobile manufacturer, headquartered in Gothenburg, Sweden. Volvo Cars manufactures and markets sport utility vehicles, station wagons, sedans, etc. It has over 15,000 employees (around 8000 employees are within research and development, for example, around 4500 engineers for a new car project).

To allow for a real context in the case study, we did not provide the subjects with a project, but asked them to select one real and non-trivial project from their work and use AADF in the context of their own projects. Therefore, we could not define up front the details of the cases, such as the set of applicable risks.

4.2.2. Case and units of analysis

According to the guidelines for conducting case studies (Runeson and Höst, 2009), a multiple-case study means that the study has multiple cases, while a single-case study refers to a case study with only one case. In this work, we treated our study as a multiple-case study, i.e., with two cases. Furthermore, the study is an embedded case study (Runeson et al., 2012), because it was conducted in two different companies of two different application domains with six architects from each company. In this work, we did not study the AAs documented by the subjects, but their opinions on the effectiveness of AADF. The architects were not only the subjects, but also the units of analysis, because we analyzed the data individually from each of the architects for the RQs.

4.2.3. Case study procedures

The case study process is composed of five tasks as the following.

T1 The researchers asked the subjects to select one real and non-trivial project from their work. The researchers prepared the materials for the case study, and collected basic information of the subjects and the selected projects through a questionnaire (see Section 4.3). The subjects and researchers reached an agreement on the selected projects.
T2 The researchers gave a tutorial to help the subjects understand the concept of AA and AADF, the tasks the subjects needed to perform, and the schedule of the workshop. Note that the correspondence rules, the essential, important, and optional elements of AADF, and the cost-benefit analysis (i.e., the guidelines on using AADF) presented in Section 3.6, were not included in the tutorial given to the subjects. After the tutorial, the subjects and researchers had a discussion about the AA concept (including both AA examples and definitions) to ensure that the subjects had a fair understanding of the AA concept. In the workshop, we did not require that the subjects understand AA exactly the same as the authors; we let them have their own understanding of AA.
T3 The subjects used an Office Excel template (implementing AADF) provided by the researchers to document AAs in their selected projects. The researchers asked the subjects to use the existing documentation of their selected projects, and their own computers in this task.

T4 The subjects exchanged the documented AAs with each other (e.g., Architect1 ⟨-⟩ Architect2, Architect3 ⟨-⟩ Architect4, and Architect5 ⟨-⟩ Architect6), and read the AADF views of their colleagues to understand the documented AAs.

T5 The researchers conducted a semi-structured interview (see Section 4.3) with each subject to get their feedback about the effectiveness of AADF in AAs Documentation. In the case of IBO Technology, the first and seventh author interviewed the subjects in parallel. In the case of Volvo Cars, the first, fourth, and fifth author interviewed the subjects in parallel. After the interview, the subjects attended a focus group (see Section 4.3) to further discuss the effectiveness of AADF in AAs Documentation according to the RQs.

Table 9 shows the relationships among tasks, who were involved, inputs and outputs of each task, and the related RQs.

4.3. Data collection procedures

Three data collection methods were used in the case study: questionnaire, interview, and focus group. We asked each subject to fill in a questionnaire (see Table 17 in Appendix C), collecting basic information of the subjects, and data related to the context of the selected projects. We interviewed (one by one) all the subjects with specific questions related to AADF (see Table 18 in Appendix C). The purpose of the focus group was to encourage the subjects to discuss the effectiveness of AADF in AAs Documentation. The researchers organized the discussion according to the four RQs. Through these three methods, we collected a number of data items (see Table 19 in Appendix C) to answer the RQs, basic information of the subjects (see Table 20 in Appendix C), and data related to the context of the selected projects (see Table 21 in Appendix C).

4.4. Analysis procedures

We used descriptive statistics to analyze quantitative answers (e.g., the size of the selected projects), and Constant Comparison (Glaser and Strauss, 1967) for qualitative answers (i.e., generating concepts and categories from the collected data to answer the RQs). The first and seventh author executed Constant Comparison through an iterative process in the IBO case. The first and fourth author did the same for the Volvo case. The codes with their relationships were refined and adapted in each iteration. The second author was consulted to resolve any inconsistency and reviewed the results of Constant Comparison for clarity. Furthermore, we used MAXQDA[5] as the tool for analyzing the qualitative data collected from both cases.

4.5. Pilot study

We conducted a pilot study with one architect in Wuhan, China. The pilot study followed the same design as the formal case study, except performing T4 (see Section 4.2.3) and the focus group (see T5 in Section 4.2.3); since there was only one subject in the pilot study, we could neither provide him with the AADF views created by others nor organize a focus group. In the end of the pilot study, we had an extra session compared with the formal case study to ask the subject for feedback on how to improve the case study. The changes from the feedback can be summarized as follows: (1) Refinement of the AADF tutorial (e.g., reducing academic terms and using more images instead of text); (2) Refinement of the Excel template (e.g., providing an explanation for each element); (3) Refinement of the interview and focus group design (e.g., the strategies used for interviews); (4) Rescheduling of the procedure of the case study (e.g., asking the subjects to fill in the questionnaire before the case study); (5) Backup plans for various emergencies and exceptions during the case study.

[5] http://www.maxqda.com/.

5. Results

This section provides first an overview of the subjects and selected projects of the two companies and subsequently presents the results of the RQs. We note that we did not add any interpretation of our own in this section.

5.1. Overview

5.1.1. IBO technology

As shown in Table 22 in Appendix D, all the subjects (6 out of 6, 100%) have at least seven years of working experience in software-intensive systems and at least five years of experience in architecting or design of software-intensive systems.

Table 23 in Appendix D shows the details of the projects selected by each subject. All the projects were non-trivial software-intensive systems from the domain of Internet of Things, and had been finished when we conducted this case study. The projects had varying duration (from two months to thirteen months), size (from 1680 to 15,000 person-hour), and lines of code (from 36,000 to 1 million). The development team size of these projects was small (4–8 persons). Most of the selected projects (3 out of 6, 50.0%) used the Waterfall Model as development process, one project (1 out of 6, 16.7%) used Iterative Development with Prototyping, one project (1 out of 6, 16.7%) used Agile Development, and one project (1 out of 6, 16.7%) used a hybrid process of the Waterfall Model and Iterative Development with Prototyping.

5.1.2. Volvo cars

As shown in Table 24 in Appendix D, all the subjects (6 out of 6, 100%) have at least seven years of working experience in software-intensive systems and at least one year of experience being involved in architecting or design of software-intensive systems.

The projects that the participants selected were mainly of two kinds: either *platform projects*: developing platforms that car models are built upon or projects for developing *new car models*. A platform project runs as long as the platform is used. The participants had a hard time to estimate the size of the projects in numbers, as they were part of an architecture group, being responsible for the electrical architecture of the complete car or platform. The software in the car is developed by around 50 different companies acting as suppliers, as well as in-house. This means that there are thousands of engineers involved, and an estimated 100 million lines of code spread over about 100 nodes in the car. Every project (except for one, which is part of a research project) followed the V-Model. Table 25 in Appendix D shows the details of the selected projects by each subject. All six projects were still running when we conducted the case study.

5.1.3. AAs documented with AADF by the subjects

This section presents the AAs documented with AADF by the subjects, as shown in Table 10. Considering the length of the paper, we only provide the names of the documented AAs. Due to confidential concerns, four subjects did not provide us with their documented AAs.

5.2. Results of RQ1

This section describes the results of RQ1, i.e., the ease of understanding the AADF viewpoints. Most subjects (5 out of 12, 41.7%)

considered that the AA Detail viewpoint is the easiest to understand. One subject stated: "*The description of each element and the examples provided in the Excel template are clear. It is more like a meta-data thing. You have to present what data that have to fill in. That was easy to understand.*" Three subjects (3 out of 12, 25.0%) mentioned that the AA Relationship viewpoint and the AA Tracing viewpoint are difficult to understand, because of the various types of relationships as well as the lack of instructions.

In the workshop, all the subjects (12 out of 12, 100%) mentioned that they could understand AADF and especially the rationale behind the AADF viewpoints. However, six subjects (6 out of 12, 50%) said that they needed further help when using AADF next time. We tried to dig deeper to identify the factors that have an impact on the ease of understanding the AADF viewpoints, and classified them as the following six factors:

Tutorial of AADF: For the best way of understanding the AADF viewpoints, half of the subjects (6 out of 12, 50%) suggested a workshop with a PowerPoint presentation. Other options proposed were a video tutorial, an online tutorial, and a presentation with examples and "homework" . Furthermore, seven subjects (7 out of 12, 58.3%) discussed the supporting documents of AADF. Providing such documents in a good quality to the subjects can help them to understand the AADF viewpoints easier. This is also a factor for creating the AADF views, which will be described in Section 5.3. Finally, four subjects (4 out of 12, 33.3%) stated that some important information was missing in the tutorial, which may negatively impact the understanding of the AADF viewpoints. They suggested elaboration on (a) the best way for learning and using AADF, (b) the benefits of AADF, (c) the purposes of each AADF viewpoint, and (d) the starting point of creating the AADF views (i.e., which view should be created first).

Examples of AADF: Eight subjects (8 out of 12, 66.7%) thought that it was important and necessary to provide more examples of using AADF. More specifically, they would like to see details about the examples (e.g., project context of the examples). Furthermore, seven subjects (7 out of 12, 58.3%) emphasized that it was better to show them how to create the AADF views with a mature and realistic industry project (e.g., decomposing a project into modules and providing examples based on each module) from their own context, instead of using toy examples.

Concepts and terms in AADF: Some concepts and terms in AADF were rather academic for the subjects, and they considered that this problem could impede their understanding of the AADF viewpoints. As one subject put it: "*We do not use some of the terms in practice (*e.g.*, the term of AA). It may confuse stakeholders when using these terms. It is better to use the terms that industry can easily understand.*"

Personal experience: Most subjects (8 out of 12, 66.7%) agreed that personal experience is an important factor for understanding the AADF viewpoints. They emphasized that there would be a significant difference in the ease of understanding AADF between junior and experienced architects, for example: "*You would need at least some experience to understand why you need AADF. There's a risk that you don't understand the need of AADF, if you haven't feel this is helpful. I know this because of my experience.*"

Resources for learning AADF: This factor refers to the resources (e.g., time) used for learning and discussing AADF. Most subjects (7 out of 12, 58.3%) considered that more resources could help stakeholders to understand the AADF viewpoints easier. One subject explained: "*It's a kind of craftsmanship and you can get better after working on it over time.*"

Practice of AADF: Three subjects (3 out of 12, 25.0%) agreed that if a stakeholder can use and practice AADF several times, especially in industry projects, she/he could understand AADF easier.

5.3. Results of RQ2

This section describes the results of RQ2, i.e., the expected effort of creating the AADF views. We asked the subjects which AADF views were expected as the most or least time-consuming to create when using AADF in their work. The results are shown in Table 11. Additionally, two subjects (2 out of 12, 16.7%) mentioned that understanding the AA concept and identifying AAs required the most effort when creating the AADF views.

We identified seven factors that can impact the effort of creating the AADF views:

Information collection: Different ways of information collection lead to different effort of creating the AADF views. Seven subjects (7 out of 12, 58.3%) thought that such effort could be reduced if they had project documents. As one subject stated: "*If we would have the whole project documentation, we can start at mapping things to find the relationships. But right now, it is hard.*" Four subjects (4 out of 12, 33.3%) suggested that architecture documents, requirements documents, risk evaluation documents, and testing documents were useful for reducing the effort of creating the AADF views. Additionally, one subject said: "*Using my memory is easier to recall the information for creating the AADF views than using documents.*" Ten subjects (10 out of 12, 83.3%) claimed that they used their memory, and five subjects (5 out of 12, 41.7%) said that they used certain documents (e.g., architecture documents) to create the AADF views. Note that the use of related documents was voluntary. In this situation, the subjects could use both their memory and project documents to create the AADF views, or only one of them. Finally, collecting different types of information for creating the AADF views also requires different effort. For example, two subjects (2 out of 12, 16.7%) considered that collecting requirements could be time-consuming, and one subject (1 out of 12, 8.3%) said that the details of an AA were difficult to collect.

Tool support: We provided an Excel template as the tool for the subjects to create the AADF views in the case study. All the subjects (12 out of 12, 100%) considered that when creating the AADF views in real life projects, a dedicated tool is needed to reduce the effort, instead of only using the Excel template. As an example, one subject mentioned: "*The Excel template is not intuitive and difficult to use and fill in the information of the AADF views.*"

AADF: The subjects mentioned seven aspects of AADF, which could impact the effort of creating the AADF views: (a) to what extent a stakeholder understands AADF (i.e., a good understanding of AADF can reduce the effort); (b) examples of AADF (e.g., examples of the AADF views with details and examples from industry projects would help stakeholders to reduce the effort); (c) characteristics of AADF (e.g., incompatible with the current processes/approaches used in projects would increase the effort); (d) ways of creating the AADF views (e.g., using AADF in the early phases of a project and iteratively creating and maintaining the AADF views during the development could reduce the effort); (e) supporting documents of AADF (this is also a factor for understanding the AADF viewpoints as described in Section 5.2); (f) practice of AADF (i.e., more practice of AADF, less effort needed); and (g) required information of AADF (i.e., which information of an AA should be documented and how to use the information).

Project context: This factor includes aspects such as project size and number of AAs (8 out of 12, 66.7%) and teamwork (4 out of 12, 33.3%). The subjects considered that large

Table 11
Effort for creating the AADF views.

Most time-consuming AADF view	Rationale	Number (%)
AA Tracing view	*The AA Tracing view is related to various software artifacts, and it requires much time to think (*e.g.*, what artifacts exist in the project, and what are the relationships between AAs and these artifacts).*	6 (50%)
AA Relationship view	*The AA Relationship view requires more thinking, and its complexity is O(n),* i.e.*, more assumptions you have, more effort you need for creating this view.*	4 (33.3%)
AA Detail view	*Several terms in the AA Detail view were confusing; when creating the view one needs not to think too much, but it requires filling in a lot of information.*	4 (33.3%)
AA Evolution view	Not mentioned	1 (8.3%)
Least time-consuming AADF view	Rationale	Number (%)
AA Detail view	*The information used to initiate the AA Detail view can be easily found.*	2 (16.7%)
AA Evolution view	*The AA Evolution view is sort of doing small steps with a low effort.*	2 (16.7%)

projects with a large number of AAs required more effort to create the AADF views than small projects, and working with multiple stakeholders collaboratively as a team could reduce such effort. As one subject said: "*We have maybe 100 decisions and maybe 400 requirements to manage. AADF was easy to understand, but maybe difficult to use, because of the size of the projects, which will make the creation of the AADF views big and complex, just because of the large number of assumptions and everything.*" Furthermore, one subject (1 out of 12, 8.3%) considered that project size and number of AAs might not be that important if the AADF views can be created and maintained iteratively.

Costs and benefits: The subjects (7 out of 12, 58.3%) argued that the effort spent on creating the AADF views mainly depends on the worthiness of using AADF in projects. As one subject put it: "*I am a little bit afraid that the effort is huge, it would be too much to detail, and just a lot of work to find the relations between everything, and to create the design work. That might be something that is no longer maintained, and could be just a mess. So you need to do some kinds of balancing between usefulness and time and effort.*"

Personal experience: Experienced architects need less effort than junior architects when creating the AADF views. Furthermore, two subjects (2 out of 12, 16.7%) considered that personal experience was related to several other aspects. They concluded that (a) if a stakeholder understands AADF well and the project has well-organized project documents; or (b) if personal experience can be embedded into AADF (e.g., similar to a reference architecture or a library of AAs), then personal experience is not important for the effort of creating the AADF views.

AAs Identification: AAs Identification may be effortless when happening during AAs Documentation. However, as reported by the subjects, identifying AAs could also be rather difficult and time-consuming: "*Because we have so much in our heads. We write requirements, design patterns, a lot of things based on so many assumptions, but we don't document assumptions, so the problem is to identify assumptions*".

5.4. Results of RQ3

This section describes the results of RQ3, i.e., the effectiveness of AADF in helping architects to identify risks in projects. 10 subjects (10 out of 12, 83.3%) agreed that using AADF could help them to identify risks, for example: "*The assumptions in themselves could be risky. Then the AA Relationship and Tracing views are also valuable because you realize how much depends on one of these assumptions, so if the assumption is not true, then you have a whole change of things that could be affected.*" Five subjects (5 out of 12, 41.7%) considered that the AA Tracing view and the AA Relationship view are the most useful for risk identification.

We also asked the subjects about the types of risk that can be identified by using AADF. The results contain architectural and design risk (4 out of 12, 33.3%), implementation risk (4 out of 12, 33.3%), management risk (3 out of 12, 25.0%), cost risk (e.g., time and money) (3 out of 12, 25.0%), product risk (2 out of 12, 16.7%), technical risk (e.g., technical debt) (2 out of 12, 16.7%), and misunderstanding risk (2 out of 12, 16.7%). As one subject put it: "*Probably we would find things that the architects assume that everyone knows, but there are many sub-system owners that are new and frequently change, so probably we would find stuff, that we thought was what everyone know, but in reality, they don't.*"

The effectiveness of using AADF to identify risks is impacted by three factors:

Personal experience: Seven subjects (7 out of 12, 58.3%) considered that more experience could be helpful to identify risks in software development when using AADF. One subject explained: "*You can't make assumptions on architecture issues, if you don't have enough experience, then you can't provide or produce relevant risks as well, so rich experience can help risk identification.*" Furthermore, one subject (1 out of 12, 8.3%) explained this factor in a different way. He thought that personal experience was not important for identifying risks, if it could be done with multiple stakeholders, because they can discuss risks based on the AADF views.

Resources: This factor refers to the resources (e.g., time) spent on risk identification using AADF. More resources may help stakeholders to identify more risks in projects. As one subject put it: "*If you review the AADF views regularly, then you could probably discover that the functional growth of the project has not been according to the predicted* (i.e.*, a risk), and we should probably start doing something earlier.*"

Project context: Various project contexts (e.g., project size and number of AAs) may increase or decrease the effectiveness of risk identification when using AADF: "*It would be difficult to identify all the risks, if there is a large number of AAs, because it is not easy to manage all the artifacts and the stakeholders may overlook some of the risks.*"

5.5. Results of RQ4

This section describes the results of RQ4, i.e., the effectiveness of AADF in helping architects to understand AAs in projects. We asked the subjects to share their created AADF views in pairs, and read their colleague's AADF views to understand the assumptions. One subject did not read the AADF views created by another subject because of other engagement during the workshop. The remaining subjects (11 out of 11, 100%) stated that they could understand the AADF views created by their colleagues. Four subjects (4 out of 12, 33.3%) explicitly said that the AADF views created by their colleagues covered all the information they expected. However, one subject (1 out of 12, 8.3%) considered that the description

Table 12
Most/least useful AADF views for AAs Understanding.

Most useful AADF view	Reasons from subjects	Number (%)
AA Detail view	*Most concepts are in the AA Detail view. If you understand them, other AADF views can be easily understood.*	7 (58.3%)
AA Tracing view	*The AA Tracing view acts as a bridge and provides an overview of AAs in a project. Most information (*e.g., *the relationships between AAs and other artifacts) can be found in this view.*	2 (16.7%)
AA Relationship view	Not mentioned	1 (8.3%)
AA Evolution view	Not mentioned	1 (8.3%)
Least useful AADF view	Reasons from subjects	Number (%)
AA Relationship view	*If a stakeholder does not understand the project, she/he cannot understand the AA Relationship view either. On the other side, the stakeholder does not need to read the AA Relationship view for understanding the AAs of the project. Furthermore, there may not be many relationships between AAs, which makes the AA Relationship view least useful.*	4 (33.3%)
AA Evolution view	Not mentioned	4 (33.3%)
AA Tracing view	*Not much information can be found if stakeholders only read the AA Tracing view.*	2 (16.7%)

and rationale in the AA Detailed view were not elaborated enough. He noted: "*For example, when describing the UHF (Ultra High Frequency) module, only providing the information of power supply and electricity is not enough. The communication is also important.*" Furthermore, we also asked the subjects which AADF views were the most or least useful for AAs Understanding. The results are shown in Table 12.

The effectiveness of using AADF to understand AAs is impacted by five factors:

- **Personal experience:** Ten subjects (10 out of 12, 83.3%) emphasized that more experience can help stakeholders to better understand AAs. As one subject put it: "*Junior architects don't understand the purpose of architecture. To understand the process, and how would the documentation of architecture emerge, you actually need to understand the architecture before you can recognize what is an assumption and what is not an assumption, because you don't have so much experience related to assumptions in the beginning.*" Furthermore, one subject connected personal experience to the quality of the AADF views: "*If the AADF views are detailed and comprehensive, personal experience (*e.g., *project experience) may not be an important factor for AAs Understanding. If the AADF views are not detailed or in a good quality, personal experience may be important for understanding the AAs in a project.*"
- **Project context:** This includes project size, number of AAs, and project data. A typical answer about the factor 'project data': "*Some information of the project is confidential, which may not be documented in the AADF views. If a stakeholder cannot see this type of information, she/he may not understand the AAs documented in the AADF views, unless she/he has been involved in the project.*"
- **AADF views:** This factor contains the elements and quality of the AADF views. Providing the AADF views in a good quality with specific information (e.g., project-related information) can improve the effectiveness of AAs Understanding. As one subject put it: "*It is better to add the background information of the system in the AA Detail view (*e.g., *the mission of the system).*"
- **Tool support:** A dedicated tool is needed to increase the effectiveness of AAs Understanding. For example, when reading the AADF views documented by another subject in Excel, it was not easy to understand all the relationships between AAs or AAs and other artifacts.
- **Terms used to describe an AA:** Stakeholders may describe and discuss AAs in different ways, which makes AAs Understanding difficult. One subject explained: "*When you make an assumption, the wording is important. 'should', 'could', 'would', 'must' are very specific words. So you have to first figure out how to express an assumption.*"

6. Discussion

We first discuss the results of the RQs, and subsequently elaborate on their potential implications for both researchers and practitioners.

6.1. Interpretation of the results

The results of RQ1 indicate that AADF can be understood in a short time (e.g., in a half day workshop), though the subjects may need further instructions when using AADF in their projects. The AA Detail viewpoint has the largest number of elements in the four viewpoints, but the subjects considered that this viewpoint was the easiest to understand. One reason could be that most of the elements in the AA Detail viewpoint are commonly used in software development (e.g., rationale, pros, and cons). On the other side, the subjects considered that the AA Relationship viewpoint and the AA Tracing viewpoint were the most difficult to understand, potentially because the relationship types in these two viewpoints are not generally used in practice.

The results of RQ2 indicate that the effort of creating the AADF views depends on various factors. It is not surprising that the AA Tracing view was considered by the subjects the most time-consuming to create, because it relates AAs to four different types of software artifacts. Traceability is important in architectural knowledge, but it usually requires significant effort to create, maintain, and evolve (Capilla et al., 2016). Additionally, understanding the relationship types in AADF and identifying the exact relationships between AAs or between AAs and other types of artifacts are not easy as mentioned by the subjects. This is also the reason that four subjects (4 out of 12, 33.3%) thought that the AA Relationship view might need the most effort to create. Finally, though the AA Detail viewpoint was considered easy to understand, it has the largest number of elements in the four viewpoints, and this is the reason that four subjects (4 out of 12, 33.3%) mentioned that creating the AA Detail view could be the most time-consuming.

The results of RQ3 indicate that AADF can help the subjects to identify risks in projects. Primarily, AA itself is a type of risk in software development, because of its uncertain nature. Another reason is that the inconsistencies between AAs or AAs and other types of artifacts could also lead to risks in projects, and this is why the AA Relationship view and the AA Tracing view were considered by the subjects the most useful for risk identification. Additionally, the subjects mentioned that AADF could help them to identify various types of risks (e.g., implementation risks and management risks), besides architectural risks. One reason is that AA itself has a broad scope (e.g., organizational AAs and business AAs), as we discussed in our survey (Yang et al., 2016).

The results of RQ4 indicate that AADF can help the subjects to understand the AAs documented by other subjects. As reported in Section 5.5, the AA Detail view was the most helpful for AAs Un-

derstanding. A probable reason is that the AA Detail view provides an overview of each AA and an entrance point to the documented AAs, while the other three AADF views are dedicated to specific aspects of AAs.

6.2. Implications for researchers

Personal experience: The impact of personal experience to AAs Documentation is an interesting topic for further investigation. There may be a significant difference between junior and experienced architects on their understanding and usage of AAs Documentation approaches. Such approaches like AADF may work well for experienced architects but may not be directly applicable for junior architects. Therefore, there is a need to investigate how personal experience impacts AAs Documentation, which can help researchers to adapt their approaches for stakeholders with different levels of experience.

Understanding of AAs Documentation approaches: When developing an approach for AAs Documentation, one needs to ensure a short learning curve so that it can be adopted in practice. We have learnt the following lessons regarding supporting practitioners to understand our framework: (1) A half-day tutorial, using partially slides and for the most part hands-on work seems to work well. Supporting stakeholders by answering questions they may have is equally important. An online tutorial might be less binding but face-to-face interaction is lost. (2) There is a trade-off between providing supporting documents to help stakeholders to understand and use the approach on the one hand, and requiring too much time from stakeholders to read this material on the other hand. We suggest a simple and intuitive set of guidelines for the documentation approach with examples and diagrams. (3) It is important to provide context information (e.g., the motivation of the documentation approach) so that stakeholders understand the "big picture" . For example, we did not elaborate the motivation and purpose of each AADF viewpoint in AAs management and software development during the presentation in the case study; this made some subjects wonder why document AAs using these four AADF viewpoints instead of other viewpoints. (4) Providing real-world instead of toy examples that are from the subjects' application domain significantly helps them to understand the documentation approach. (5) When designing the training material, reducing the number of academic concepts and terms is a major concern. However, it may be hard to completely avoid using them. For example, we found that several concepts and terms (e.g., the term AA itself) in AADF were rarely used at the two companies. Clearly explaining the concepts and terms of the proposed documentation approach with examples can alleviate this problem.

Tool support was identified as one important factor for the effort of creating the AADF views and was related to the effectiveness of AAs Understanding. For example, after creating the AA Detail view, it would be helpful if the tool can automatically generate the other three types of AADF views. We believe that tool support is also important for other AAs Documentation approaches, especially when using them in industry. We suggest that researchers develop and provide dedicated tools for their AAs Documentation approaches instead of using general tools (e.g., Office Excel), and consider integrating the tools with existing development tools; this would make it easier to employ the documentation approaches with existing development practices and reduce the manual effort of AAs Documentation.

Guidelines on using AADF: Since the case study aimed to evaluate the effectiveness of AADF for AAs Documentation (see Section 4.1), no RQs are related to the guidelines of using AADF (e.g., the essential, important, and optional elements of AADF, see Section 3.6). We suggest that researchers need to further evaluate the guidelines on using AADF (e.g., correctness and appropriateness).

Practical applicability of AAs management: Though AAs are important in software development, they are usually not well managed in projects. One potential reason is that the return on investment of AAs management is unknown. Therefore, the practical applicability of AAs management (e.g., the practical applicability of AADF) needs further evaluation and evidence. We suggest that researchers collect more data regarding especially the return on investment for AAs management (e.g., using AADF to manage AAs).

6.3. Implications for practitioners

Personal experience: It is not surprising that personal experience was identified as an important factor for understanding AADF, creating the AADF views, identifying risks in projects, and understanding the assumptions documented by other practitioners; all the four aspects are based on the knowledge of practitioners. We believe that personal experience has a significant impact on AAs Documentation. For example, junior architects may not even be aware of the AAs made in software development. Suggestions to alleviate this problem are to discuss with experienced architects about AAs as well as their documentation, and practice AAs Documentation in projects.

Project context: Different project contexts can impact the effort of documenting AAs, the effectiveness of identifying risks, and understanding AAs in projects. For example, the subjects in the case study mentioned that their resources (e.g., time) were limited for AAs Documentation and they needed to balance the resources spent on various tasks (e.g., architecture design) in software development, which is a common situation in industry. We suggest practitioners to be aware of the costs and benefits of documenting AAs in their projects, and integrate AAs Documentation into their own development process in an appropriate way. Furthermore, for large projects, we suggest practitioners to document AAs in the early phases of software development, and maintain the documented AAs iteratively.

Quality of AAs Documentation: Practitioners need to ensure the quality of AAs Documentation, as that can impact several aspects of AAs management (e.g., the effectiveness of AAs Understanding between various stakeholders). Ding et al. collected and classified twelve quality attributes of software documents through a systematic literature review (Ding et al., 2014). Specifically, according to our experience in industry, we suggest that practitioners need to pay more attention to understandability, clarity, unambiguity, retrievability, consistency, and correctness of their documented AAs.

Expression of AAs: As suggested by the subjects, there should be a consistent way to express AAs when using a documentation approach such as AADF. Providing rules for expression of AAs can facilitate AAs management (e.g., AAs Communication), but (1) it also leads to increased effort for understanding and using these rules, and (2) the rules may not be compatible with the existing development practices employed in the company (e.g., requirements expression). Therefore, there is a need to reach an agreement on expressions of AAs within a project team before using an AAs Documentation approach.

7. Threats to validity

The threats to the validity of the case study are presented in this section according to the guidelines proposed by Runeson and Höst (Runeson and Höst, 2009). Note that internal validity is not discussed, because this work does not study causality.

Construct validity reflects to what extent the research questions and the studied operational measures are consistent

Table 13
Factors for applying AADF in AAs Documentation.

Factor	Subfactor	RQs
AADF	Tutorial of AADF (Approaches of tutorial; Supporting documents of AADF in tutorial; Necessary information in tutorial)	RQ1
	Examples of AADF (Number of examples; Quality of examples; Types of examples)	RQ1, RQ2
	Practice of AADF	
	Concepts and terms in AADF	RQ1, RQ4
	Understanding of AADF	RQ2
	Characteristics of AADF	
	Approaches of creating the AADF views	
	Supporting documents of AADF	
	Required information of AADF	
	AADF views (Elements of the AADF views; Quality of the AADF views)	RQ4
Personal experience		RQ1, RQ2, RQ3, RQ4
Resources (e.g., time)	Resources for learning AADF	RQ1
	Resources for risk identification	RQ3
Information collection	Approaches of information collection	RQ2
	Types of information to be collected	
Costs and benefits		RQ2
AAs Identification		RQ2
Tool support		RQ2, RQ4
Project context	Project size	RQ2, RQ3, RQ4
	Number of AA	
	Others	

(Runeson and Höst, 2009). A potential threat is that the subjects tried to use AADF without a fair understanding of the AA concept. To mitigate this threat, after the tutorial, the subjects and researchers had a discussion about both the AA examples and definition to ensure its proper understanding. Furthermore, whether the collected data can answer the RQs is also a threat. To reduce this threat, we iteratively refined the RQs and the data collection procedures; we used both interview and focus group to collect data for the RQs.

External validity concerns the generalization of the findings (Runeson and Höst, 2009). This case study was conducted in two companies from two domains in two countries; we argue that the results are applicable in organizations with similar context. However, the results cannot be generalized to other contexts (e.g., other domains) and replication of this case study is one way to reduce this threat.

Reliability focuses on whether the study would yield the same results when other researchers replicate it (Runeson and Höst, 2009). We performed a pilot study to refine the case study design, and reduced the ambiguities in the execution of the case study. The protocol of the case study has been reviewed by the authors iteratively, and also by an external reviewer who is not familiar with AADF, to mitigate the threat of bias in the case study design. The whole process of the workshops were recorded through audio recording devices to reduce the threat of information vaporization and erosion. Furthermore, two authors conducted the analysis of qualitative data in parallel for each case to reduce the threat of bias in data analysis.

8. Conclusions and future work

As an important type of architectural knowledge, AAs usually remain implicit and undocumented, which leads to many problems in software development. To this end, we proposed a framework to explicitly and systematically document AAs in projects based on the identified concerns of stakeholders. AADF is the first approach that follows the guidelines proposed in ISO/IEC/IEEE 42010 (ISO, 2011) for AAs Documentation. AADF provides a template to document the details of each AA, supports stakeholders to get an overview of the relationships between AAs, trace AAs to other software artifacts, and monitor the evolution of AAs in software development.

To evaluate the effectiveness of AADF in AAs Documentation, we conducted a case study with two industrial cases of two companies (i.e., IBO Technology and Volvo Cars) from two domains (Internet of Things and Automotive industry respectively) in two countries (i.e., China and Sweden respectively). The results of the case study indicate that (1) AADF can be understood in a short time (i.e., a half day workshop); (2) the AA Evolution view requires the least time to create, followed by the AA Detail view and the AA Relationship view; (3) AADF can help stakeholders to identify risks and understand AAs in projects (the AAs documented by other stakeholders); and (4) the ease of understanding AADF, the effort of creating the AADF views, and the effectiveness of risk identification and AAs Understanding using AADF are related to various factors as shown in Table 13. Adjusting these factors in an appropriate way can facilitate the usage of AADF and further benefit the projects.

It is important to investigate the long-term utility of the documentation created by applying AADF and the issues of the AAs documented when the architecture evolves. These can only be evaluated during the lifecycle of a project and not with a half-day workshop. This work is a preliminary evaluation for the short-term use of AADF; we consider a long-term use of AADF in a longitudinal study as future work. Other future directions include (1) investigating the various factors concerning the applicability of AADF identified in this study (e.g., how the factors impact the effort of creating the AADF views); (2) adapting AADF to address additional AA concerns; (3) investigating which types of AAs should be documented; (4) developing a dedicated tool to support the application of AADF in industry projects; and (5) designing a systematic AAs management process, which can be integrated in the architecting process.

Acknowledgements

This work is partially sponsored by the NSFC under Grant No. 61472286 and the Ubbo Emmius scholarship program by the University of Groningen. This work is also partially supported by the Vinnova FFI projects Next Generation Electronic Architecture and Next Generation Electronic Architecture step 2. We would like to thank the architects in AA concerns selection, the participants of the case study from IBO Technology and Volvo Cars, and the architect who participated in the pilot study. We would also like to thank Christian Manteuffel for reviewing an earlier version of the paper.

Appendix A. Examples of the AADF viewpoints

Tabel 14, Tabel 15.

Table 14
An example of the Architectural Assumption Evolution viewpoint.

Iteration	Description
i	the project manager assumed that "*The management subsystem would be deployed in an (secure-enough) internal environment*" (Assumption: AA1). Then the architect assumed that "*It might not be necessary to consider the external security (such as broken access control and cross-site scripting) of the system*" (Assumption: AA2), because of AA1.
j	the project manager considered "*Deploying the system directly on Internet*", and AA1 and AA2 became invalid.
k	AA1 was modified to "*It is uncertain whether the management sub-system would be deployed directly on Internet*", and AA2 was modified to "*External security of the system may need to be considered*".
l	the customer confirmed AA1 and AA2 turning them into requirements: "*The system would be deployed directly on the Internet*" (Requirement: R1), and "*The external security of the system should be considered*" (Requirement: R2).
m	according to R2, the architect assumed that "*Data validation and data encryption might be enough to support the external security of the system*" (Assumption: AA3).
n	AA3 was turned into a decision: "*Data validation and data encryption were used for the external security of the system*" (Architectural Design Decision: ADD1).

Table 15
An example of the Architectural Assumption Detail viewpoint.

ID	AA1
Name	Response time of the system
Description	The response time of the system should be within 2 seconds
State	Valid and Added
Rationale	The assumption is made based on the architect's experience and knowledge.
Pros	High usability
Cons	Extra effort (such as testing, hardware)
Invalidity reason	N/A
Stakeholder	Related stakeholders: YC, FT, JY, PX, SC, TY, WL, LX, YY Affected by HT

Appendix B. Definitions of the context elements in AADF

Table 16.

Table 16
Definitions of the context elements in AADF.

Name	Definition
Stakeholder	*Individuals, teams, organizations, etc., who have an interest in a system* (ISO, 2011).
Artifact	*A piece of information that is produced, modified, or used by a process* (Kroll and Kruchten, 2003).
Risk	*Characterized by the probability of an event that will result in a negative impact plus a characterization of the negative impact on a project* (Bourque and Fairley, 2014).
Requirement	*The needs and constraints placed on a software product that contribute to the solution of real-world problems* (Bourque and Fairley, 2014).
Architectural design decision	*A restructuring effect on the components and connectors that make up the architecture, design rules imposed on the architecture and resulting system as a consequence of the design decision, design constraints imposed on the architecture, and a rationale explaining the reasoning behind the decision* (Bosch, 2004).
Component	*An independent unit with well-defined interfaces and dependencies that can be composed and deployed independently* (Bourque and Fairley, 2014).

Appendix C. Case study design

Table 17, Table 18, Table 19, Table 20, Table 21.

Table 17
Questionnaire about the basic information of the subjects and the selected projects.

Question	Type of Answers
What is your experience (in years) working in software-intensive systems?	Integer > = 0
How long (in years) have you worked in the current company?	Integer > = 0
What is your experience (in years) in software architecting?	Integer > = 0
Have you ever received any professional training (i.e., excluding higher education) related to software architecture or software design?	Yes / No
Can you describe the selected project?	Free text
What is the duration of the selected project?	Months
What is the team size of the selected project?	Persons
What is the size of the selected project?	Person-hours
How many lines of code of the selected project?	Integer > = 0
What is the development process employed in the selected project?	Free text (e.g., Agile Development and Iterative Development)

Table 18
Questions for interview.

RQ1: How easy is it to understand the AADF viewpoints?
IQ1. To what extent are you able to use AADF in your projects without any help (easy, moderate, or difficult)?
IQ1.1 Which aspects of AADF are you unsure about?
IQ1.2 Which aspects would you require further training?
IQ2. How easy is it for you to understand the AADF viewpoints (easy, moderate, difficult)?
IQ2.1 If it is moderate or difficult, what made it difficult for you to understand the AADF viewpoints?
IQ2.2 Which AADF viewpoint is the easiest for you to understand?
IQ2.3 Why do you think this AADF viewpoint is the easiest for you to understand?
IQ2.4 Which AADF viewpoint is the most difficult for you to understand?
IQ2.5 Why do you think this AADF viewpoint is the most difficult for you to understand?
IQ3. Have you encountered any difficulties to understand the AADF viewpoints?
IQ3.1 What are the difficulties you encountered to understand the AADF viewpoints?
IQ4. What do you think is the best way to understand the AADF viewpoints (e.g., watching a video)?
IQ4.1 How important is the quality of the learning material for understanding the AADF viewpoints?
IQ5. How important is the time spent for understanding the AADF viewpoints?
IQ5.1 How much time do you expect to spend on understanding the AADF viewpoints?
IQ6. How important is the architecting experience for understanding the AADF viewpoints?
IQ6.1 How does the architecting experience influence understanding the AADF viewpoints?
IQ7. Are there any other factors that would influence your understanding to the AADF viewpoints?
RQ2: What is the effort for creating the AADF views?
IQ8. When using Office Excel templates to create the AADF views, is the effort acceptable to you?
IQ8.1 Which AADF view took you the most time to create?
IQ8.2 What makes it difficult for you to create this view?
IQ8.3 Which AADF view took you the least time to create?
IQ8.4 Why this view is the easiest for you to create?
IQ8.5 If you use another tool (e.g., pen and paper) to create the AADF views, are the views that took you the most time and the least time the same as using Office Excel templates?
IQ9. Did you collect the information from existing project documentation to create the AADF views or just created the views from what you can remember?
IQ9.1 How easy is it for you to collect the required information from e.g., documents or your memory for creating the AADF views?

(*continued on next page*)

Table 18 (*continued*)

IQ9.2 Which information (e.g., state and the related requirements) is the easiest for you to collect when creating the AADF views?
IQ9.3 Why do you think these types of information are the easiest to collect?
IQ9.4 Which information is the most difficult to collect when creating the AADF views?
IQ9.5 Why do you think these types of information are the most difficult to collect?
IQ10. Do you like to create the AADF views using Office Excel templates?
IQ10.1 If no, which tool would you prefer to use to reduce the effort for creating the AADF views?
IQ10.2 If yes, how easy is it to create the AADF views with Office Excel templates?
IQ11. How important is the quality of the existing project documentation for creating the AADF views?
IQ11.1 Which documentation is the most important for creating the AADF views?
IQ11.2 Why do you think the xx documentation is the most important for creating the AADF views?
IQ12. How important is the architecting experience for creating the AADF views?
IQ12.1 How does the architecting experience influence you for creating the AADF views?
IQ13. How important is the project context (e.g., project domain, project size, and team size) for creating the AADF views?
IQ13.1 How does the project context influence you for creating the AADF views?
IQ14. Are there any other factors that would influence your effort for creating the AADF views?
RQ3: Does AADF effectively help architects to identify risks in projects?
IQ15. Were you able to identify any risks by creating the AADF views?
IQ15.1 If yes, can you describe the identified risks?
IQ16. Which AADF views can help you to identify risks?
IQ16.1 Why the AADF views can help you to identify risks?
IQ17. Which types of risks (e.g., architectural risks and management risks) can be identified through using the AADF views?
IQ17.1 How did you identify xx risks (e.g., architectural risks) when using the AADF views?
IQ18. How important is the time spent for identifying risks?
IQ18.1 How does the time spent influence identifying risks?
IQ19. How important is the architecting experience for identifying risks?
IQ19.1 How does the architecting experience influence identifying risks?
IQ20. Are there any other factors that would influence your identification of risks?
RQ4: Does AADF effectively help architects to understand AA in projects?
IQ21. Were you able to understand the AA from the views documented by your colleague?
IQ22. Which AADF view is most useful for you to understand AA and why?
IQ23. Which AADF view is least useful for you to understand AA and why?
IQ24. Do the created AADF views cover all the information you need in understanding AA of a project?
IQ24.1 What details are easy to understand when reading the AADF views?
IQ24.2 What details are not easy or are missing in the AADF views?
IQ25. How important is the time spent for understanding AA?
IQ25.1 How does the time spent influence your understanding of AA?
IQ26. How important is the architecting experience for understanding AA?
IQ26.1 How does the architecting experience influence your understanding of AA?
IQ27. How important is the project context (e.g., project domain, project size, and team size) for understanding AA?
IQ27.1 How does the project context influence your understanding of AA?
IQ28. Are there any other factors that would influence your understanding of AA?

Table 19
Data items collected for answering the RQs.

Interview question	Scale type	Unit	Range	RQ
IQ1	N/A	N/A	Easy / Moderate / Difficult	RQ1
IQ1.1	N/A	N/A	Text	RQ1
IQ1.2	N/A	N/A	Text	RQ1
IQ2	N/A	N/A	Easy / Moderate / Difficult	RQ1
IQ2.1	N/A	N/A	Text	RQ1
IQ2.2	N/A	N/A	AADF viewpoints	RQ1
IQ2.3	N/A	N/A	Text	RQ1
IQ2.4	N/A	N/A	AADF viewpoints	RQ1
IQ2.5	N/A	N/A	Text	RQ1
IQ3	N/A	N/A	Yes / To some extent / No	RQ1
IQ3.1	N/A	N/A	Text	RQ1
IQ4	N/A	N/A	Watching a video / Reading documents / Training by a coach / Discussing with team members	RQ1
IQ4.1, IQ5 - IQ7	N/A	N/A	Text	RQ1
IQ8	N/A	N/A	Yes / To some extent / No	RQ2
IQ8.1	N/A	N/A	AADF views	RQ2
IQ8.2	N/A	N/A	Text	RQ2
IQ8.3	N/A	N/A	AADF views	RQ2
IQ8.4	N/A	N/A	Text	RQ2
IQ8.5	N/A	N/A	Text	RQ2
IQ9	N/A	N/A	Only documents / Only memory / Both	RQ2
IQ9.1	N/A	N/A	Text	RQ2
IQ9.2	N/A	N/A	State / Rationale / Pros / Cons / Invalidity reason / Stakeholder / Plan / Related architectural assumptions / Related requirements / Related architectural design decision / Related system components / Related risks / History	RQ2
IQ9.3	N/A	N/A	Text	RQ2
IQ9.4	N/A	N/A	State / Rationale / Pros / Cons / Invalidity reason / Stakeholder / Plan / Related architectural assumptions / Related requirements / Related architectural design decision / Related system components / Related risks / History	RQ2
IQ9.5	N/A	N/A	Text	RQ2
IQ10	N/A	N/A	Yes / To some extent / No	RQ2
IQ10.1	N/A	N/A	Tools	RQ2
IQ10.2	N/A	N/A	Easy / Moderate / Difficult	RQ2
IQ11 - IQ14	N/A	N/A	Text	RQ2
IQ15	N/A	N/A	Yes / To some extent / No	RQ3
IQ15.1	N/A	N/A	Text	RQ3
IQ16	N/A	N/A	AADF views	RQ3
IQ16.1	N/A	N/A	Text	RQ3
IQ17	N/A	N/A	Text	RQ3
IQ17.1	N/A	N/A	Text	RQ3
IQ18 - IQ20	N/A	N/A	Text	RQ3
IQ21	N/A	N/A	Yes / To some extent / No	RQ4
IQ22	N/A	N/A	AADF views	RQ4
IQ23	N/A	N/A	AADF views	RQ4
IQ24	N/A	N/A	Yes / No	RQ4
IQ24.1	N/A	N/A	State / Rationale / Pros / Cons / Invalidity reason / Stakeholder / Plan / Related architectural assumptions / Related requirements / Related architectural design decision / Related system components / Related risks / History	RQ4
IQ24.2	N/A	N/A	Text	RQ4
IQ25 - IQ28	N/A	N/A	Text	RQ4

Table 20
Data items related to subjects.

Data item	Scale type	Unit	Range
Time the subjects have worked in software-intensive systems	Ratio	Years	>=0
Time the subjects have worked in the current company	Ratio	Years	>=0
Time the subjects have been involved in architecting or design of software-intensive systems	Ratio	Years	>=0
Architecture-related training	N/A	N/A	Yes / No

Table 21
Data items related to the selected projects.

Data item	Scale type	Unit	Range
Description of the selected project	N/A	N/A	Text
Duration of the selected project	Ratio	Months	>=0
Team size of the selected project	Ratio	Persons	>=0
Size of the selected project	Ratio	Person-hours	>=0
Lines of code of the selected project	Ratio	Lines	>=0
Development process employed in the selected project	N/A	N/A	Development processes

Appendix D. Overview of the subjects and selected projects

Table 22, Table 23, Table 24, Table 25.

Table 22
Experience of the subjects at IBO technology.

Data item	P1	P2	P3	P4	P5	P6
Experience in working in software-intensive systems (Years)	13	15	7	10	8	18
Employment in the current company (Years)	9	10	5	6	5	7
Experience in architecting or design of software-intensive systems (Years)	10	5	6	8	7	15
Architecture training (Yes / No)	No	No	No	Yes	No	No

Table 23
Basic information of the selected projects at IBO Technology.

Project	Duration (Months)	Team size (Persons)	Project size (Person-hour)	Lines of code	Development process
P1	3	4	2016	36,000	Waterfall Model
P2	3	6	3024	52,000	Agile Development
P3	6	5	5220	85,000	Iterative Development with Prototyping technique
P4	13	8	15,000	500,000	Waterfall Model
P5	2	5	1680	100,000	Waterfall Model
P6	12	5	10,800	1000,000	A hybrid process of the Waterfall Model and Iterative Development with Prototyping technique

Table 24
Experience of the subjects at Volvo Cars.

Data item	P1	P2	P3	P4	P5	P6
Experience in working in software-intensive systems (Years)	20	27	30	7	12	20
Employment in the current company (Years)	3	20	1	1	5	3
Experience in architecting or design of software-intensive systems (Years)	10	20	30	1	5	3
Architecture training (Yes / No)	No	No	No	No	No	No

Table 25
Basic information and description of the selected projects at Volvo Cars.

Project	Duration (Months)	Development process
P1	120	V-Model
P2	36	V-Model
P3	120	V-Model
P4	48	Hybrid process
P5	36	V-Model
P6	36	V-Model

References

Adolph, S., Hall, W., Kruchten, P., 2011. Using grounded theory to study the experience of software development. Empirical Softw. Eng. 16 (4), 487–513.

Boehm, B.W., 1991. Software risk management: principles and practices. IEEE Softw. 8 (1), 32–41.

Bosch, J., 2004. Software architecture: The next step. In: Proceedings of the 1st European Workshop on Software Architecture (EWSA). St Andrews, UK, pp. 194–199.

Bourque, P., Fairley, R.E., 2014. Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0. IEEE Computer Society Press.

Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Babar, M.A., 2015. 10 years of software architecture knowledge management: practice and future. J. Syst. Softw. 116 (6), 191–205.

Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Babar, M.A., 2016. 10 years of Software Architecture Knowledge Management: Practice and Future. J. Syst. Softw. 116 (6), 191–205.

Ding, W., Liang, P., Tang, A., van Vliet, H., 2014a. Knowledge-based approaches in software documentation: A systematic literature review. Inf. Softw. Technol. 56 (6), 545–567.

Ding, W., Liang, P., Tang, A., van Vliet, H., Shahin, M., 2014b. How do open source communities document software architecture: An exploratory survey. In: Proceedings of the 19th International Conference on Engineering of Complex Computer Systems (ICECCS). Tianjin, China, pp. 136–145.

Fleming, C.H., Leveson, N., 2015. Integrating systems safety into systems engineering during concept development. In: INCOSE International Symposium, 25, pp. 989–1003.

Garlan, D., Allen, R., Ockerbloom, J., 2009. Architectural mismatch: Why reuse is still so hard. IEEE Softw. 26 (4), 66–69.

Garlan, D., 2010. Software engineering in an uncertain world. In: Proceedings of the 18th FSE/SDP Workshop on Future of Software Engineering Research (FoSER). Santa Fe, New Mexico, USA, pp. 125–128.

Glaser, B.G., Strauss, A.L., 1967. The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Publishing, New York.

Haley, C.B., Laney, R.C., Moffett, J.D., Nuseibeh, B., 2006. Using trust assumptions with security requirements. Requirements Eng. 11 (2), 138–151.

Hesse, T.M., Paech, B., 2013. Supporting the collaborative development of requirements and architecture documentation. In: Proceedings of the 3rd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks). Rio de Janeiro, Brazil, pp. 22–26.

Heesch van, U., Avgeriou, P., Hilliard, R., 2012. A documentation framework for architecture decisions. J. Syst. Softw. 85 (4), 795–820.

ISO. ISO/IEC/IEEE Std 42010-2011, Systems and Software Engineering – Architecture Description, 2011.

Jansen, A., Avgeriou, P., van der Ven, J.S., 2009. Enriching software architecture documentation. J. Syst. Softw. 82 (8), 1232–1248.

Kroll, P., Kruchten, P., 2003. The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison-Wesley Professional.

Lago, P., van Vliet, H., 2004. Observations from the recovery of a software product family. In: Proceedings of the 3rd International Conference on Software Product Lines (SPLC). Boston, MA, USA, pp. 214–227.

Lago, P., van Vliet, H., 2005. Explicit assumptions enrich architectural models. In: Proceedings of the 27th International Conference on Software Engineering (ICSE). St. Louis, Missouri, USA, pp. 206–214.

Lehman, M.M., Ramil, J.F., 2001. Rules and tools for software evolution planning and management. Ann. Softw. Eng. 11 (1), 15–44.

Lewis, G.A., Mahatham, T., and Wrage L. Assumptions management in software development. Technical Report, CMU/SEI-2004-TN-021, 2004.

Mamun, M.A.A., Hansson, J., 2011. Review and challenges of assumptions in software development. In: Proceedings of the 2nd Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS). Vienna, Austria.

M.A.A. Mamun, M. Tichy, and J. Hansson. Towards Formalizing Assumptions on Architectural Level: A Proof-of-Concept. Research Report, University of Gothenburg, 2012.

Miranskyy, A., Madhavji, N., Davison, M., Reesor, M., 2005. Modelling assumptions and requirements in the context of project risk. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE). Paris, France, pp. 471–472.

Ordibehesht, H., 2010. Explicating critical assumptions in software architectures using AADL Master Thesis. University of Gothenburg.

Ostacchini, I., Wermelinger, M., 2009. Managing assumptions during agile development. In: Proceedings of the 4th Workshop on SHAring and reusing architectural Knowledge (SHARK). Vancouver, Canada, pp. 9–16.
Roeller, R., Lago, P., van Vliet, H., 2006. Recovering architectural assumptions. J. Syst. Softw. 79 (4), 552–573.
Rost, D., Naab, M., Lima, C., von Flach Garcia Chavez, C., 2013. Software architecture documentation for developers: A survey. In: Proceedings of the 7th European Conference (ECSA). Montpellier, France, pp. 72–88.
Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng. 14 (2), 131–164.
Runeson, P., Host, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons.
Steingruebl, A., Peterson, G., 2009. Software assumptions lead to preventable errors. IEEE Security Privacy 7 (4), 84–87.
Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Babar, M.A., 2010. A comparative study of architecture knowledge management tools. J. Syst. Soft. 83 (3), 352–370.
Tirumala, A., Crenshaw, T., Sha, L., Baliga, G., Kowshik, S., Robinson, C., Witthawaskul, W., 2005. Prevention of failures due to assumptions made by software components in real-time systems. ACM SIGBED Rev. 2 (3), 36–39.
Uchitel, S., Yankelevich, D., 2000. Enhancing architectural mismatch detection with assumptions. In: Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS). Edinburgh, UK, pp. 138–146.
Van Landuyt, D., Joosen, W., 2014. Modularizing early architectural assumptions in scenario-based requirements. In: Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE). Grenoble, France, pp. 170–184.
Van Landuyt, D., Truyen, E., Joosen, W., 2012. Documenting early architectural assumptions in scenario-based requirements. In: Proceeding of the Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA). Helsinki, Finland, pp. 329–333.
William, M., Donnelly, J., 2006. The Research Methods Knowledge Base. Atomic Dog.
Yang, C., Liang, P., 2014. 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.
Yang, C., Liang, P., Avgeriou, P., 2016. A survey on software architectural assumptions. J. Syst. Soft. 113 (3), 362–380.
Yang, C., Liang, P., and Avgeriou P. Assumptions and their management in software development: A systematic mapping study. (under review). 2017.
Yang, C., Liang, P., Avgeriou, P., Eliasson, U., Heldal, R., Pelliccione, P., Bi., T.. Replication Package for AADF.
Yin, R.K., 2003. Case study research. Design and Methods, 3rd edition Sage, London.
Zschaler, S., Rashid, A., 2011. Aspect assumptions: a retrospective study of AspectJ developers' assumptions about aspect usage. In: Proceedings of the 10th International Conference on Aspect-Oriented Software Development (AOSD). Porto de Galinhas, Brazil, pp. 93–104.

Chen Yang is a PhD candidate in the State Key Lab of Software Engineering (SKLSE), School of Computer Science, Wuhan University, China and the Software Engineering and Architecture (SEARCH) research group at the University of Groningen, the Netherlands. He is working in the research field of software architecture. Before becoming a PhD student, he had worked as a senior software engineer and architect in industry for four years. He has published several articles in peer-reviewed international journals and conferences.

Peng Liang is Professor of Software Engineering in the State Key Lab of Software Engineering (SKLSE), School of Computer Science, Wuhan University, China. His research interests concern the areas of software architecture and requirements engineering. He is a Young Associate Editor of Frontiers of Computer Science, Springer. He has published more than 90 articles in peer-reviewed international journals, conferences and workshop proceedings, and books.

Paris Avgeriou is Professor of Software Engineering at the University of Groningen, the Netherlands where he has led the Software Engineering research group since September 2006. Before joining Groningen, he was a post-doctoral Fellow of the European Research Consortium for Informatics and Mathematics. He is an Associate Editor for IEEE Software and sits on the editorial board of Springer Transactions on Pattern Languages of Programming (TPLOP). He has co-organized several international conferences and workshops (mainly at ICSE). His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution, patterns and link to requirements.

Ulf Eliasson is employed as an industrial PhD at Volvo Cars. His research education is at the Department of Computer Science and Engineering, Chalmers and University of Gothenburg, Sweden. At Volvo Cars he is part of a group working with processes and tools for software development, continuous integration, and quality assurance. Before becoming a PhD student, he has worked as a software engineer in industry for two years. He has published several articles in peer-reviewed international conferences.

Rogardt Heldal is Professor at the Bergen University College, Bergen, Faculty of Engineering and Business Administration, and he is also Associate Professor (on leave) at the Chalmers University of Technology and University of Gothenburg, Sweden, Department of Computer Science and Engineering. He got his PhD in 2000 at the Chalmers University of Technology. His research topics are mainly in software engineering, software architectures modelling, software processes and empirical studies. He has co-authored more than 70 publications in journals and international conferences and workshops in this theme. He has been on the program committees for several conferences, and is a reviewer for top journals in the software engineering domain. He is active in International and National projects. He is an ISERN member. In his research activity he has collaborated with several industries such as Volvo Cars, Volvo AB, Ericsson, Grundfos, Axis communication and Saab.

Patrizio Pelliccione is an associate professor in the Department of Computer Science and Engineering at Chalmers University of Technology and the University of Gothenburg. He got his PhD in 2005 at the University of L'Aquila (Italy) and from February 1, 2014 he is Docent in Software Engineering, title given by the University of Gothenburg. His research topics are mainly in software engineering, software architectures modelling and verification, autonomous systems, and formal methods. He has co-authored more than 100 publications in journals and international conferences and workshops in these topics. He has been on the program committees for several top conferences, and is a reviewer for top journals in the software engineering domain. He is very active in European and National projects. In his research activity he has collaborated with several industries such as Volvo Cars, Volvo AB, Ericsson, Jeppesen, Axis communication, Thales Italia, Selex Marconi telecommunications, Siemens, Saab, TERMA, etc. Contact him at patrizio.pelliccione@gu.se; www.patriziopelliccione.com.

Tingting Bi is a PhD candidate in the State Key Lab of Software Engineering (SKLSE), School of Computer Science, Wuhan University, China and a visiting PhD candidate at the Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia. She received her bachelor and master degrees in computer science and technology from the Harbin University of Science and Technology, China. Her current research interests include software architecture and machine learning. She has published several articles in peer-reviewed international journals.