Architectural Assumptions and Their Management in Industry – An Exploratory Study

Chen Yang[1,2], Peng Liang[1(✉)], Paris Avgeriou[2], Ulf Eliasson[3,4], Rogardt Heldal[4], and Patrizio Pelliccione[4]

[1] State Key Lab of Software Engineering, School of Computer Science, Wuhan University, Wuhan 430072, China
liangp@whu.edu.cn

[2] Department of Mathematics and Computing Science, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands

[3] Volvo Cars, Volvo Jacobs Väg, 405 31 Gothenburg, Sweden

[4] Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, 412 96 Gothenburg, Sweden

Abstract. As an important type of architectural knowledge, architectural assumptions should be well managed in projects. However, little empirical research has been conducted regarding architectural assumptions and their management in software development. In this paper, we conducted an exploratory case study with twenty-four architects to analyze architectural assumptions and their management in industry. In this study, we confirmed certain findings from our previous survey on architectural assumptions (e.g., neither the term nor the concept of architectural assumption is commonly used in industry, and stakeholders may have different understandings of the architectural assumption concept). We also got five new findings: (1) architects frequently make architectural assumptions in their work; (2) the architectural assumption concept is subjective; (3) architectural assumptions are context-dependent and have a dynamic nature (e.g., turning out to be invalid or vice versa during their lifecycle); (4) there is a connection between architectural assumptions and certain types of software artifacts (e.g., requirements and design decisions); (5) twelve architectural assumptions management activities and four benefits of managing architectural assumptions were identified.

Keywords: Architectural assumption · Architectural assumptions management · Case study

1 Introduction

The concept of assumption in software engineering is not new. Various types of assumptions have been investigated in the software engineering literature, such as requirement assumptions [1], architectural assumptions [10], and code-level assumptions [2], each focusing on a different aspect of the software development lifecycle. Architectural assumptions (AA[1]) are an important type of architectural knowledge in both architecting and software development in general [5]. Similar to the definition of "assumption" in Oxford English Dictionary[2] and Merriam-Webster[3], we define AA as: architectural knowledge taken for granted, or accepted as true without evidence. The essence of the AA concept is "uncertainty": stakeholders are not (completely) certain regarding various aspects of architectural knowledge, including correctness, impact, importance, suitability, etc. As an example, a stakeholder may assume that "*the number of users (visitors) of the system would be around 1 million per day*". When the uncertainty of an AA is eliminated, the AA can be removed or transformed to other types of software artifacts (e.g., a design decision).

AA that are not well managed (and thus remain implicit AA or become invalid AA) can lead to a multitude of problems in software development [5]. As an example of such a problem, consider architectural misunderstanding: stakeholders may misunderstand an architectural design decision, because they are not aware of the AA behind this decision. Another example is undetected risks: one essential characteristic of assumptions is uncertainty, which may lead to risks in projects, especially if AA remain implicit.

Little empirical research has been conducted regarding the notion of AA as well as their management. In this paper, we conducted an exploratory case study with twenty-four architects to analyze AA and their management in industry. The results confirm certain findings from our previous survey on AA [5]: (1) neither the term nor the concept of AA is commonly used in industry, and stakeholders may have different understandings of the AA concept; (2) AA are not only important in architecting, but also of paramount importance in software development as they span the whole software development lifecycle; (3) there is a lack of approaches, tools, and guidelines for AA management, and there are certain challenges in managing AA. Furthermore, we got five new findings: (1) architects frequently make AA in their work; (2) the AA concept is subjective; (3) AA are context-dependent and have a dynamic nature (e.g., turning out to be invalid or vice versa during their lifecycle); (4) there is a connection between AA and certain types of software artifacts (e.g., requirements and design decisions); (5) twelve AA management activities and four benefits of managing AA were identified.

The rest of this paper is structured as follows: Sect. 2 describes related work on AA and their management. Section 3 introduces the case study design. Section 4 presents the results of the case study, while Sect. 5 discusses the findings. Section 6 describes the threats to the validity of the case study, and Sect. 7 concludes this work along with future research directions.

[1] AA in this paper is singular as well as plural based on the context in which it is used.

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

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

2 Related Work

Garlan et al. [11] treated AA as an important factor that causes architectural mismatch. The authors suggested that guidelines should be provided for documenting AA (e.g., how to integrate AA Documentation into architecture documentation). The authors further suggested several approaches (e.g., architecture views and description languages) and techniques (e.g., XML) to support AA Documentation.

Lago and van Vliet [10] distinguished AA from requirements and constraints as the reasons for architectural design decisions that are arbitrarily taken based on personal experience and knowledge. An assumption meta-model was proposed to document these assumptions in an explicit way. The authors classified AA into three types: (1) managerial assumptions, (2) organizational assumptions, and (3) technical assumptions. Roeller et al. [9] classified AA into four types: (1) implicit and undocumented (the architect is unaware of the assumption, or it concerns tacit knowledge), (2) explicit but undocumented (the architect takes a decision for a specific reason), (3) explicit and explicitly undocumented (the reasoning is hidden), (4) explicit and documented (this is the preferred, but often exceptional, situation). The authors also proposed an approach (RAAM – Recovering Architectural Assumption Method) for AA Recovery from five sources in development (e.g., source code and documentation).

Van Landuyt et al. [12] discussed a specific type of AA (i.e., early AA), which are made by requirements engineers in the early phases of development (e.g., requirements elicitation). The authors highlighted the necessity of the documentation of early AA. In their subsequent work, Van Landuyt and Joosen [13] introduced a metamodel and an instantiation strategy to document early AA based on quality attribute scenarios and use cases.

Ordibehesht [14] argued that implicit and invalid AA 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 AA.

Mamun and Hansson [15] conducted a literature review on assumptions in software development. In their review, the authors identified 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. [16] proposed to use Alloy language to document AA in software development.

Ostacchini and Wermelinger [17] proposed a lightweight approach to manage AA in agile development, and summarized four main tasks of AA management from existing literature: (1) recording new assumptions, (2) monitoring assumptions regularly, (3) searching for assumptions, and (4) recovering past assumptions. The authors used the taxonomy of AA proposed by Lago and van Vliet in [10].

In our previous work [18], we focused on AA and their documentation in agile development, and proposed a simplified conceptual model for AA with a lightweight approach for AA Documentation. Furthermore, we surveyed 112 practitioners to investigate the practice of AA in software development [5]. The results of the survey show that most AA are kept implicit due to the lack of documentation; the lack of specific approaches and tools is the major challenge (reason) of (not) identifying and documenting AA.

To the best of our knowledge, there are currently no exploratory case studies regarding AA and their management in software development from architects' perspective. The aforementioned studies mostly focus on proposing and evaluating approaches for AA management, while this study aims to explore how architects perceive AA as well as the existing activities, practices, tools, challenges, and benefits of AA management. Moreover, this work is a follow-up from our survey on AA [5]; we detail the comparison between the survey and this study in Sect. 3.2.

3 Case Study

We followed the guidelines proposed by Runeson and Höst [6] to design and report on this case study.

3.1 Goal and Research Questions

The goal of the case study, formulated using the Goal-Question-Metric approach [3], is to **analyze** AA and their management **for the purpose** of characterization **with respect to** understanding AA and activities, practices, tools, challenges, and benefits of AA management **from the point of view of** architects **in the context of** software development in industry. The research questions (RQs) of this study according to the goal are formulated as follows:

RQ1: How do architects perceive AA?

Stakeholders may perceive AA differently [5]. This RQ intends to explore how architects understand the concept of AA through definitions and examples, as well as characteristics of AA and potential relationships between AA and other software artifacts (e.g., requirements).

RQ2: What are the activities, practices, tools, challenges, and benefits of AA management?

As evidenced in our systematic mapping study on assumptions and their management in software development [4], assumptions management is comprised of a set of assumptions management activities (e.g., Assumptions Making) and supported by various practices and tools. Furthermore, AA management leads to certain benefits in software development but also has challenges. This RQ aims at helping researchers and practitioners to get a practical understanding of AA management in software development.

3.2 Case and Units of Analysis

Our case study explores a phenomenon (managing AA) in a real context, by asking each subject to select one non-trivial software project from their work and to manage AA in the context of the selected project. Note that we did not study the AA managed by the subjects, but their opinions on AA and their management. Therefore, we treat this study as a multiple and holistic case study [8]: each architect is both a case and a unit of analysis. Furthermore, this case study aims at exploring new insights of AA and their management as well as generating new ideas for further research, so it is an exploratory case study [6].

This case study follows up on our earlier work [5]: a survey with 112 practitioners on AA and their identification and documentation in software development. Compared to that survey, there are several key differences with the current study: (1) We only used questionnaire for data collection to answer the RQs in our survey, while this case study employs both interview and focus group for data collection (two different data sources help in improving the validity of the study) to answer the RQs; (2) The subjects in the survey were practitioners in software development, including various roles, such as project manager and designer, while the subjects of this study were architects; (3) The subjects in the survey were asked to fill in a questionnaire and give their opinions, while the subjects in this study received a tutorial on AA and their management and managed AA in their own projects as practice; (4) The scope of this study is broader, as it not only extends from AA Identification and Recording (i.e., the survey) to AA management in general (i.e., this study), but also included several new aspects, such as characteristics of AA and relationships between AA and other software artifacts (e.g., requirements).

3.3 Data Collection and Analysis

We conducted five workshops (half day per workshop, including a half-hour tutorial on AA and their management) in Beijing and Shenzhen, China and Gothenburg, Sweden with twenty-four architects from ten companies and different domains (e.g., Internet of Things and Automotive Industry) to collect data.

Three data collection methods were used in the case study: questionnaire, interview, and focus group. We asked each subject to fill in a questionnaire to collect their background information. We interviewed all the subjects (one by one, 30 min per subject) with specific questions related to the RQs. We conducted five focus groups (30 min per focus group per workshop) according to the RQs.

We used descriptive statistics to analyze quantitative answers (i.e., background information of the subjects), and Constant Comparison [7] for qualitative answers (i.e., generating concepts and categories from the collected data to answer the RQs). Constant Comparison (the core of the Grounded Theory approach) is a systematic approach used for qualitative analysis, and a continuous process for verifying the generated concepts and categories [7]. In this case study, Constant Comparison was iteratively performed, and the codes and their relationships were refined in each iteration. Table 1 shows the relationships among the data collection methods, data analysis methods, and RQs. Furthermore, we used MAXQDA[4] to analyze the qualitative data.

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

Table 1. Relationships among the data collection methods, data analysis methods, and RQs

Data collection method	Data analysis method	RQs
Questionnaire	Descriptive statistics	Background information
Interview	Constant comparison	RQ1, RQ2
Focus group	Constant comparison	RQ1, RQ2

4 Results

4.1 Subjects Experience and Projects Information

The experience of the subjects in software-intensive systems and architecting is generally classified in three levels as shown in Fig. 1. Most of the subjects (22 out of 24, 91.7%) have at least five years of experience in software-intensive systems, and 15 subjects (out of 24, 62.5%) have at least five years of experience in architecting. We also asked the subjects whether they had architecture-related training (excluding higher education). Four subjects (out of 24, 16.7%) claimed that they had such training experience.

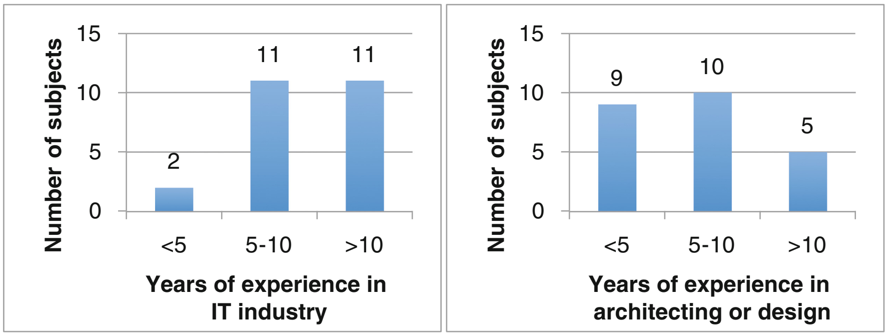

Fig. 1. Years of experience in software-intensive systems and architecting of the subjects

Furthermore, the twenty-four subjects managed AA in twenty-eight projects, i.e., there are four subjects that each of them had managed AA in two of their projects. The duration, team size, and lines of code of the projects are shown in Fig. 2. Note that two subjects did not provide us the lines of code of their projects (two projects) because the projects were in progress when we conducted the workshops.

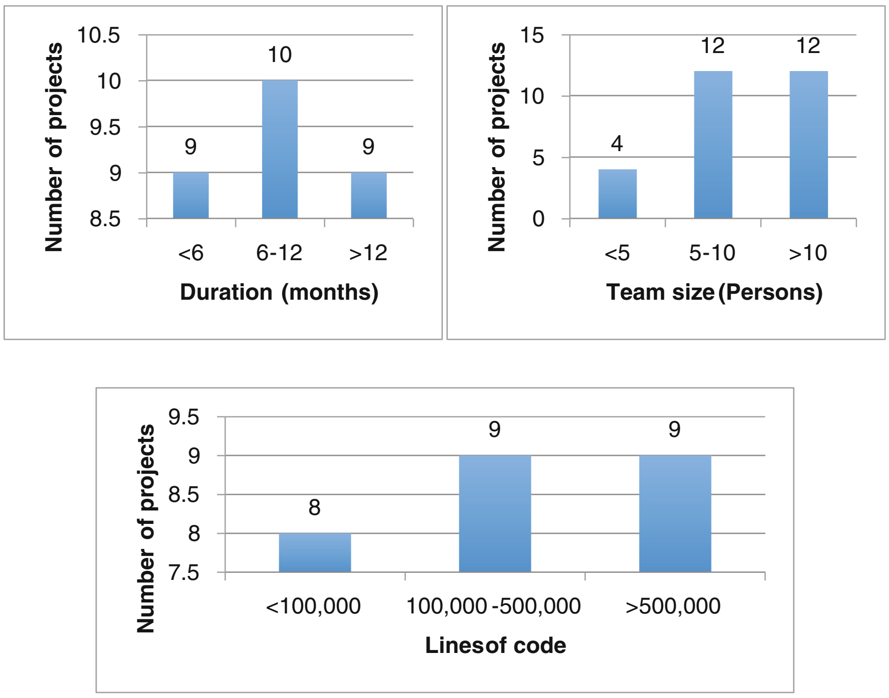

Fig. 2. Information of the projects used by the subjects in the case study

4.2 Results of RQ1 – Perception of AA

Term and Concept of AA

The results show that neither the term nor the concept of AA is commonly used in industry, although the subjects admitted that they frequently made AA in their work. As one subject put it: "*I had something in my thinking about architecture work that transformed into an assumption, but I missed that, and assumption is a very good word to use here.*"

Moreover, the results support that stakeholders may have different understandings of the AA concept. One of the most intensive arguments between the subjects concerned the nature of AA. As one subject understood: "*When we develop systems, we have various flows. AA are like uncertain key points in these flows.*" As another subject stated: "*If a statement is made based on personal experience without being able to prove that it is feasible or has no risks, it is an AA.*"

Finally, the subjects agreed that AA are not only important in architecting, but also of paramount importance in software development, and the influence of AA is through the entire development process, instead of only during the architecting or design phase. As one subject explained: "*AA are made from requirements engineering. When architecting, we make new AA, and the existing AA would evolve. I think AA can influence the whole project lifecycle as well as product lifecycle.*"

AA Examples

The subjects provided more than twenty AA examples based on their own understanding. We present two examples: "*As we can't foresee the functional growth, we need to have a scalable network topology, which means it should be possible to add network segments without changing the architecture, assuming that the actual design of the network topology is not part of the architecture.*" Another assumption: "*There would not be any need for a high speed bus inside the vehicle dynamics area until 2019 or later. This is an explicit assumption. We actually wrote it down.*"

Characteristics of AA

The subjects mentioned that the AA concept is subjective (e.g., whether something is architecturally-significant or whether an information is an assumption). This is a reason that stakeholders may have a different understanding of the AA concept, and can make different AA according to their own understanding.

Furthermore, AA are context-dependent, i.e., they can be different depending on context (e.g., project context). As one subject put it: "*If I have assumption1 and assumption2, and they have a relationship in project 1. For project 2, they don't have any relationship. So the relationship between these two AA is context-dependent on the project.*"

Moreover, the subjects talked about the dynamic nature of AA, i.e., AA can evolve over time. This means that a valid AA can turn out to be invalid or vice versa, as well as an AA can transform to another type of software artifact or vice versa during its lifecycle: "*AA may transform to another artifact, or something that is not an AA in the first place, but change to an AA. The transformation between AA and other artifacts should be bi-directional.*"

Finally, the subjects agreed that relationships between two AA could be zero, one, or multiple. As one subject stated: "*In some way, two types of relationships (e.g., "is caused by" and "constrains") could coexist between two AA.*"

AA and Other Software Artifacts

The subjects agreed that AA are not independent in software development; instead, there is a connection between AA and other software artifacts. As one subject mentioned: "*I can see that a lot of things are based on assumptions. When we have a new project, we could start with some basic or important assumptions that we have, and base design patterns or requirements on those AA.*"

The subjects listed several artifacts related to AA, including requirements, design decisions, design patterns, design models, components, and risks. As one subject stated: "*Some requirements we have are actually AA, and we have strategies based on assumptions, so it could be good to acknowledge that.*" As another subject put it: "*How does an AA evolve to a requirement? The dependencies between assumptions and things like that are the most important. We could connect AA to decisions, requirements, and components.*"

Summary of RQ1

We summarize the aforementioned results of RQ1: (1) Neither the term nor the concept of AA is commonly used in industry. (2) The subjects (architects) frequently made AA in their work. (3) Stakeholders may have different understandings of the AA concept.

(4) AA are important in both architecting and software development. (5) AA span the whole software development lifecycle. (6) The AA concept is subjective. (7) AA are context-dependent and have a dynamic nature. (8) There is a connection between AA and certain types of software artifacts.

4.3 Results of RQ2 – AA Management

AA Management Activities

By considering the assumptions management activities identified and summarized in our mapping study [4], we found that all the twelve assumptions management activities were mentioned by the subjects in the case study: AA Making, Description, Evaluation, Maintenance, Tracing, Monitoring, Communication, Understanding, Reuse, Recovery, Searching, and Organization. These AA management activities include both the activities the subjects used in their projects and the activities the subjects considered important or difficult. For example, the subjects agreed that making AA (including both identifying existing AA and making new AA) was hard for them. Furthermore, AA Making, Description, Evaluation, Maintenance, Tracing, and Monitoring were the most frequently discussed AA management activities by the subjects in the case study.

Practices and Tools Used for AA Management

The subjects did not employ any general AA management process or approaches in their work, but they used certain practices and tools for AA management activities. We listed all the practices and tools used by the subjects in Table 2. Note that the tools are listed without a mapping to the specific AA management activities because the subjects did not provide that information.

Challenges of AA Management

Furthermore, we identified a set of challenges regarding AA management in software development as shown in Table 3.

Benefits of AA Management

Finally, we identified a set of benefits of AA management in software development as shown in Table 4.

Summary of RQ2

We summarize the aforementioned results of RQ2: (1) Twelve AA management activities were identified. (2) No systematic approaches were used by the subjects for AA management. (3) Nine practices without guidelines were used by the subjects in five AA management activities. (4) All the tools used by the subjects for AA management are general in software development. (5) There are eight challenges and four benefits identified in managing AA.

Table 2. Practices and tools used for AA management

AA management activity	Practice	Tools
AA making	Making AA in requirements engineering	MS PowerPoint[a]; MS Visio[b]; MS Word[c]; MS Project[d]; Enterprise architect[e]; Rational Software Architect Designer[f]; SmartDraw[g]; Origin[h]; PowerDesigner[i]; ProcessOn[j]; internal tools
	Making AA in architecting	
	Making AA using brainstorming	
AA description	Describing AA in documents (e.g., design documents)	
	Describing AA in models (e.g., architecture models)	
AA evaluation	Evaluating AA in architecture evaluation	
	Evaluating AA in software testing	
AA maintenance	Involving customers and users in the discussions of maintaining AA	
AA communication	Face-to-face communication of AA	

[a]https://products.office.com/en/powerpoint
[b]https://products.office.com/en-us/visio/flowchart-software
[c]https://products.office.com/en-us/word
[d]https://products.office.com/en-us/project/project-and-portfolio-management-software
[e]http://www.sparxsystems.com/
[f]http://www-03.ibm.com/software/products/en/ratsadesigner
[g]https://www.smartdraw.com/
[h]http://www.originlab.com/
[i]http://www.powerdesigner.biz/EN/
[j]https://www.processon.com/

Table 3. Challenges of AA management

Challenge	Description
Understanding of AA	Neither the concept nor the term of AA was commonly used by the subjects. It is challenging to understand the AA concept and term: "*It was difficult to get started. You need to figure out getting to the mode of understanding, which was the trickiest part.*" Furthermore, stakeholders may have different understandings of the AA concept, which could cause inconsistency in AA management
AA management activities	One of the most important challenges is how to conduct individual AA management activities in software development, i.e., there is a lack of specific approaches for AA management. For example, considering AA Making and Description, one subject put it: "*We write a lot of things based on so many assumptions, but we don't document. The problem is actually how to catch the assumptions because we have so much in our heads.*"

(*continued*)

Table 3. (*continued*)

Challenge	Description
Tools	There is a lack of tools for AA management, which should be able to deal with different project context (e.g., an AA is valid in a project, but invalid in another project), support not only AA Description, but also other AA management activities (e.g., Making and Evaluation), have good quality of outputs, and support automation or semi-automation. As one subject mentioned: "*We have maybe 100 decisions and 400 requirements, and we have different projects, which have the same assumptions in different validation states, so the tool must handle these situations and be very scalable.*"
Lack of data	There is a lack of data (e.g., empirical data) regarding AA and their management in existing projects, which makes conducting AA management activities (e.g., AA Reuse and Evaluation) difficult. As one subject put it: "*If you have data regarding AA from 10,000 projects, you can generate an AA model to analyze AA in future projects.*"
Integration	AA are related to other software artifacts, such as requirements. Therefore, there is a challenge to integrate AA and their management with existing software development processes, approaches, tools, etc. An example given by one subject: "*In embedded systems, flowchart diagram is the most important, and AA management should be compatible with flowchart diagrams, when introducing AA in embedded systems.*"
Project context	AA management depends on project context (e.g., resources, complexity, development processes employed, and application domain). Thus, an AA management approach may work in one project, but not work in another project. As one subject explained: "*For some projects AA management may grow to be a huge unwieldy thing that everyone is afraid of, and you need to maintain, because no one else touches that.*"
Experience	AA management requires certain experience (project experience, architecting experience, etc.). One challenge is how to mitigate the gap between junior and experienced stakeholders regarding AA management. As one subject stated: "*Experienced architects understand AA management better than junior architects. Experienced architects can make more reasonable AA, while junior architects may not even know what AA they have or need to make. You can't just go out on the street and pick up the first guy, and say: you will be the architect!*"
Stakeholders	There is a lack of guidance regarding who should be involved in AA management. As one subject mentioned: "*AA are related to various aspects of software development, and AA management should be teamwork.*"

Table 4. Benefits of AA management

Benefit	Description
Being aware of AA and related problems	The most intuitive benefit of managing AA is to make AA explicit, i.e., stakeholders become aware of the AA made in projects. This can further help stakeholders to be aware of and avoid potential problems (e.g., risks) caused by implicit or invalid AA. As one subject put it: "*It would definitely be good for us to acknowledge that we have assumptions in the first place and to work with them to some extent.*"
Improving traceability	AA are not independent in software development, but intertwined with various software artifacts, e.g., design decisions. Management of AA can help to trace AA to other artifacts. As one subject stated: "*The benefits of managing AA are that you can see what decisions have been made based on assumptions, and know why you made some decisions.*"
Facilitating maintenance and handover	AA are usually implicit, and intertwined with various software artifacts. Management of AA can make AA explicit, and prevent knowledge vaporization in software development, which can further facilitate maintenance and handover within projects. As one subject mentioned: "*AA management enriches software knowledge, which helps to maintain, for example, architecture or source code.*"
Reducing costs	Invalid AA may lead to problems such as inappropriate architecture design, and consequently increase costs (e.g., additional development effort) of a project. AA management aims at reducing invalid AA and thus reduces cost. As one subject explained: "*Invalid AA would make the system that is based on the architecture more expensive than it needs to be.*"

5 Discussion

5.1 Interpretation of RQs Results

Interpretation of the Results of RQ1

There is an obvious paradox: on the one hand, neither the term nor the concept of AA was commonly used in industry; on the other hand, the subjects frequently made AA in their work. One reason could be that the subjects were not aware of the AA when they made them. The most probable reason however is that AA were not treated as first class entities in software development. Instead, AA were considered as, for example, a type of constraint or rationale of other software artifacts (e.g., design decisions).

Stakeholders may have a different understanding of the AA concept. One reason could be that the AA concept is subjective, as the subjects mentioned. Furthermore, AA are context-dependent (e.g., project context). This indicates that, for example, an AA can be valid in one project but invalid in another project depending on the context. As mentioned by the subjects, one potential reason is that AA are related to various artifacts. For example, an AA can be caused or constrained by a requirement.

Therefore, if the requirement changes in another project, the AA may also change (e.g., from valid to invalid).

Moreover, AA have a dynamic nature, i.e., AA can evolve over time. This means that a valid AA can turn out to be invalid or vice versa, but also that an AA can transform to another type of software artifact or vice versa. A potential reason for the former (i.e., bi-directional changes between valid and invalid) is the context-dependent characteristic of AA. For the latter (i.e., bi-directional transformation between AA and other types of software artifacts), the reason could be that AA are inherently uncertain: once the uncertainty of an AA is eliminated, this AA transforms to another type of artifact; an artifact, such as a design decision, may also become uncertain and thus turn into an AA.

Finally, as stated by the subjects, AA are not only important in architecting, but also of paramount importance in software development; its lifecycle is throughout the whole software development lifecycle. One reason could be that AA are related to various software artifacts, such as requirements, design decisions, components, and risks. Another reason, as the subjects explained, is that AA management is teamwork, involving different stakeholders, instead of only architects.

Interpretation of the Results of RQ2

The results of RQ2 confirm the twelve assumptions management activities identified and summarized in our systematic mapping study [4]. Furthermore, AA Making, Description, Evaluation, Maintenance, Tracing, and Monitoring got the most attention by the subjects. One reason may be that the subjects considered these six activities as the primary AA management activities in software development; if AA management is employed in development, these activities are more likely to take place or get more attention than other activities.

In this study, we did not find any particular approaches the subjects used for AA management, which is consistent with the findings of our survey on AA [5]. Though the subjects suggested several practices for managing AA, these practices are general without any elaborated guidelines. We also found several tools for AA management used by the subjects. However, all of them are general software development tools, and thus do not specifically focus on AA management.

Furthermore, besides the lack of guidelines, approaches, tools, resources, etc. for AA management, which have also been discussed in our survey on AA [5], we found several other challenges of AA management in software development, including “Lack of data”, “Integration”, and “Project context” as elaborated in Table 3. For example, as mentioned by the subjects, there is a lack of data (e.g., empirical data) regarding AA and their management in existing projects. This is potentially because stakeholders do not make AA explicit and document them in a systematic way. The lack of existing data regarding AA from projects is also a reason that impedes conducting individual AA management activities in software development.

Finally, we identified four benefits of AA management in software development. However, we argue that these benefits are not for free, as they depend on certain conditions (e.g., related to specific AA management activities), and there is always a tradeoff between benefits and costs. For example, AA Tracing helps to improve traceability between AA and other software artifacts in software development, but the effort needed for establishing traces could be prohibitive.

5.2 Implications for Researchers

AA in Empirical Studies

The AA concept is subjective, and stakeholders may have different understandings of the AA concept. When conducting empirical studies regarding AA and their management, these nuances need to be taken into account during the study design. Especially when evaluating related approaches or tools for AA management in empirical studies, researchers need to make a decision: allowing their subjects to have their own understanding of AA or enforcing a consistent definition.

AA Management

On the one hand, AA are important in both architecting and software development. On the other hand, there are various challenges regarding AA and their management that need to be addressed in software development, as we identified in the case study. There is a clear need to develop, for example, dedicated approaches and tools, as well as well-designed practices and guidelines (e.g., when to manage AA or what AA should be managed in software development) for AA management.

Furthermore, the reasons that AA are usually not-well managed could be various: for example, the challenges listed in Table 3, or the return on investment is rather limited. We suggest that researchers collect evidence regarding the return on investment for AA management. Moreover, not-well managed AA can lead to a multitude problems in software development. There is a need for researchers to identify these problems from both literature and empirical studies. This could motivate spending extra effort on AA management.

Finally, since AA are intertwined with various types of software artifacts and their lifecycle spans the whole software development lifecycle, there is a possibility for further research regarding integrating AA management into existing software development approaches (e.g., decision-centric architecture evaluation approaches).

5.3 Implications for Practitioners

Treating AA as First Class Entities

AA are important in both architecting and software development, as they span the whole software development lifecycle. However, practitioners (e.g., architects) frequently make AA in their daily work, without always being aware what AA they made. We advocate treating AA as first class entities in software architecting as well as in software development, and integrating AA management with existing processes, approaches, tools, etc. in software development.

Understanding of AA

Understanding of AA (e.g., the AA concept) is usually an issue in AA management. We suggest that practitioners in a project should at least reach an agreement on, for example, what AA are, as well as how to manage them.

Teamwork
AA management is teamwork. Although, according to our systematic mapping study [4], in the context of software design, architects and designers are the major stakeholders in assumptions management, we encourage practitioners with different roles (e.g., project manager) being involved in managing AA. For example, practitioners can evaluate AA as a team, instead of only letting architects perform AA Evaluation.

Experience
AA management requires certain experience (including project experience and architecting experience). In general, experienced practitioners understand AA management deeper and perform it better than junior practitioners. We encourage discussions regarding AA and their management between practitioners with different levels of experience to alleviate this issue.

6 Threats to Validity

The threats to the validity of this case study are presented in this section according to the guidelines proposed by Runeson and Höst [6]. Note that internal validity is not discussed in this paper because this work does not study causality.

Construct validity
reflects to what extent the research questions and the studied operational measures are consistent [6]. A potential threat concerns whether the collected data can answer the RQs. To reduce this threat, we iteratively refined the RQs and the data collection procedures. To improve the validity of the case study we used both interviews and focus groups for the data collection.

External validity
concerns the generalization of the findings [6]. The subjects were architects from various companies and domains, and with different levels of working experience in software intensive-systems and architecting; we argue that the results are representative for practitioners with a similar background. However, the results may not be generalized to other contexts (e.g., project managers and programmers); 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 [6]. We performed a pilot study to refine the case study design (e.g., the interview questions), and reduced the ambiguities in the execution of the case study. The protocol of the case study was reviewed by the researchers iteratively, and also by eight external reviewers, to mitigate the threat of bias in the design of the case study. The whole process of the case study was recorded through audio recording devices to reduce the threat of information vaporization. Furthermore, two authors conducted Constant Comparison through MAXQDA in parallel to reduce the threat of bias in the qualitative data analysis.

7 Conclusions and Future Work

As an important type of architectural knowledge, little empirical research has been conducted regarding AA and their management in software development. In this paper, we conducted an exploratory case study with twenty-four architects to analyze AA and their management in industry.

In this study, we confirmed certain findings from our previous survey on AA [5], including (1) neither the term nor the concept of AA is commonly used in industry, and stakeholders may have different understandings of the AA concept; (2) AA are not only important in architecting, but also of paramount importance in software development as they span the whole software development lifecycle; (3) there is a lack of approaches, tools, and guidelines for AA management, and there are certain challenges in managing AA. Furthermore, we had five new findings: (1) architects frequently make AA in their work; (2) the AA concept is subjective; (3) AA are context-dependent and have a dynamic nature (e.g., turning out to be invalid or vice versa during their lifecycle); (4) there is a connection between AA and certain types of software artifacts (e.g., requirements and design decisions); (5) twelve AA management activities and four benefits of managing AA were identified.

Our next steps are: (1) developing approaches for AA management, and particularly a general AA management process in software development; (2) developing practices and guidelines for AA management to address, for example, the identified challenges in the case study; and (3) developing a dedicated tool for AA management. Note that our intention is not to develop a standalone tool, but a tool integrated with existing software development tools (e.g., a plug-in of the existing tools).

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 participants of the case study, and the architects who participated in the pilot study.

References

1. Haley, C.B., Laney, R.C., Moffett, J.D., Nuseibeh, B.: Using trust assumptions with security requirements. Requir. Eng. **11**(2), 138–151 (2006)
2. Lehman, M.M., Ramil, J.F.: Rules and tools for software evolution planning and management. Annal. Soft Eng. **11**(1), 15–44 (2001)
3. Basili, V., Caldiera, G., Rombach, D.: The Goal Question Metric Approach. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering. Wiley, New York (1994)
4. Yang, C., Liang, P., Avgeriou, P.: Assumptions and their management in software development: A systematic mapping study (under review)
5. Yang, C., Liang, P., Avgeriou, P.: A survey on software architectural assumptions. J. Syst. Softw. **113**(3), 362–380 (2016)
6. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**(2), 131–164 (2009)

7. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Publishing, New York (1967)
8. Runeson, P., Host, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and Examples. Wiley, New york (2012)
9. Roeller, R., Lago, P., van Vliet, H.: Recovering architectural assumptions. J. Syst. Softw. **79**(4), 552–573 (2006)
10. Lago, P., van Vliet, H.: Explicit assumptions enrich architectural models. In: Proceedings of the 27th International Conference on Software Engineering (ICSE), St Louis, Missouri, USA, pp. 206–214 (2005)
11. Garlan, D., Allen, R., Ockerbloom, J.: Architectural mismatch: Why reuse is still so hard. IEEE Softw. **26**(4), 66–69 (2009)
12. Van Landuyt, D., Truyen, E., Joosen, W.: 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 (2012)
13. Van Landuyt, D., Joosen, W.: 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 (2014)
14. Ordibehesht, H.: Explicating critical assumptions in software architectures using AADL. Master thesis, University of Gothenburg (2010)
15. Mamun, M.A.A., Hansson, J.: Review and challenges of assumptions in software development. In: Proceedings of the 2nd Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS), Vienna, Austria (2011)
16. Mamun, M.A.A., Tichy, M., Hansson, J.: Towards formalizing assumptions on architectural level: a proof-of-concept. Research report, University of Gothenburg (2012)
17. Ostacchini, I., Wermelinger, M.: Managing assumptions during agile development. In: Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK), Vancouver, BC, Canada, pp. 9–16 (2009)
18. Yang, C., Liang, P.: Identifying and recording software architectural assumptions in agile development. In: Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE), Vancouver, Canada, pp. 308–313 (2014)