University of Groningen

A survey on software architectural assumptions

Yang, Chen; Liang, Peng; Avgeriou, Paris

Published in:
Journal of Systems and Software

DOI:
[10.1016/j.jss.2015.12.016](https://doi.org/10.1016/j.jss.2015.12.016)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2016

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):

Yang, C., Liang, P., & Avgeriou, P. (2016). A survey on software architectural assumptions. *Journal of Systems and Software*, 113, 362-380. <https://doi.org/10.1016/j.jss.2015.12.016>

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

A survey on software architectural assumptions

Chen Yang^{a,b}, Peng Liang^{a,*}, Paris Avgeriou^b

^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

ARTICLE INFO

Article history:

Received 1 July 2015

Revised 3 December 2015

Accepted 11 December 2015

Available online 19 December 2015

Keywords:

Software architecture

Architectural assumption

Industrial survey

ABSTRACT

Context: Managing architectural assumptions (AA) during the software lifecycle, as an important type of architecture knowledge, is critical to the success of projects. However, little empirical evidence exists on the understanding, identification, and recording of AA from the practitioners' perspective.

Objective: We investigated the current situation on (1) how practitioners understand AA and its importance, and (2) whether and how practitioners identify and record AA in software development.

Method: A web-based survey was conducted with 112 practitioners, who use Chinese as native language and are engaged in software development in China.

Results: The main findings are: (1) AA are important in both software architecting and development. However, practitioners understand AA in different ways; (2) only a few respondents identified and recorded AA in their projects, and very few approaches and tools were used for identifying and recording AA; (3) the lack of specific approaches and tools is the major challenge (reason) of (not) identifying and recording AA.

Conclusions: The results emphasize the need for a widely accepted understanding of the AA concept in software development, and specific approaches, tools, and guidelines to support AA identification and recording.

© 2015 Elsevier Inc. All rights reserved.

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 (Haley et al., 2006), architectural assumptions (Roeller et al., 2006), and code-level assumptions (Lehman and Ramil, 2001), which focus on different aspects of the software development life-cycle. Stakeholders (e.g., developers, architects, and maintainers) frequently make assumptions during their daily work for various purposes (e.g., about the interpretation of requirements or characteristics of input data) (Lewis et al., 2004).

This paper focuses on architectural assumptions (AA¹), the assumptions concerning architecture (a detailed discussion of AA definition based on the survey results can be found in Section 5.1). Software Architecture (SA) represents “the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution” (ISO/IEC/IEEE Std 42010-2011, 2011). In SA, AA is an important type

of architectural knowledge (Roeller et al., 2006, Van Landuyt and Joosen, 2014). We define AA as “a statement about uncertain architectural knowledge”. For example, an architect made an educated guess that the number of users (visitors) of the system would be around 1 million per day. Apparently, this number is only an estimation; the architect cannot be sure about the accurate number until the system is deployed and operated. Therefore this assumption is architectural knowledge that contains uncertainty and will remain an assumption until this uncertainty is eliminated. This paper aims to refine and elaborate the definition of AA according to the practitioners' perspective on AA.

Architecture assumptions, like other kinds of assumptions in software engineering, have a dynamic nature: the context of the project (e.g., business environment) as well as the software itself changes over time, making formerly valid assumptions invalid, which results in unsatisfactory systems (Lewis et al., 2004). For example, the architect assumed that it is not necessary to consider the external security of the system (such as broken access control or cross-site scripting), because the system would be deployed in an (secure-enough) internal environment. However, during the development, the requirement changes in a way that external access is necessary. In this case, the initial assumption that “external security is not a concern” is invalidated. Of course, an assumption can also be invalid in the first place. This is often the case due to

* Corresponding author. Tel.: +86 27 68776137; fax: +86 27 68776027.

E-mail address: pliangeng@gmail.com, liangp@whu.edu.cn (P. Liang).

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

insufficient information while making an assumption. For example, the architect may assume that Java is a suitable language for the project without being aware of the developers' low proficiency in Java.

AA usually remain implicit and undocumented during architecting (Roeller et al., 2006) and this is a major issue because of their dynamic nature. Without explicitly identifying and recording AA, they tend to vaporize, leaving the development team unaware of which AA had been made, their implications on the architecture, and their status (i.e., whether they are still valid or have become invalid). Vaporization of AA in a project leads to two problems:

- (1) Architectural misunderstanding and mismatch. For example, stakeholders may misunderstand the architectural design decisions (ADDs), because they are not aware of the AA. Furthermore, AA incompatible with architecture design elements may result in architectural mismatch, e.g., mismatch between components or connectors (Garlan et al., 2009). Subsequently this may lead to design violations and low architecture quality.
- (2) The development team needs to spend considerable time and effort to understand and maintain the architecture.

Another problem related to AA is that practitioners may not have a common understanding of the AA concept, leading to inconsistent treatment of AA. For example, one stakeholder may consider an AA as a requirement, while others may treat the same AA as an ADD (Roeller et al., 2006). The different treatments of AA can result in misunderstandings in the communication between various stakeholders, which may impede the development of the project.

Little empirical research has been conducted in the field of AA, especially in AA identification and recording. To this end, we conducted a web-based survey with 112 practitioners, who use Chinese as native language (Chinese respondents are easily accessible to two of the researchers of this survey) and are engaged in software development in China, from a number of different industries and application domains. The objectives of the survey are the following:

- (1) To identify the practitioners' perception of AA and the importance of AA in software development. We aimed at exploring how practitioners understood the AA concept without biasing them, so we did not use our own definition of AA in the survey. Note that "the AA term" is not the same as "the AA concept". A term is a label used to express a concept (i.e., semantics) through a word or phrase (i.e., syntax), while a concept conveys the meaning of a term.
- (2) To collect the approaches and tools used to identify and record AA as well as the practical challenges associated with AA identification and recording.
- (3) To find out about the reasons for not identifying and recording AA in industry practice.

The results of our survey suggest the need for establishing a widely accepted understanding of the AA concept in software development, and specific approaches, tools, and guidelines to support AA identification and recording.

The rest of this paper is structured as follows: Section 2 discusses related work. Section 3 describes the research approach in detail. Section 4 presents the survey results. Sections 5 and 6 discuss the findings and threats to validity of the survey respectively. Section 7 concludes this survey with future work directions.

2. Related work

Related work on assumptions in software engineering and AA is discussed in this section, covering definitions and classifications of

assumptions, as well as methods of managing them (such as identifying and recording assumptions).

2.1. Assumptions in software engineering

Tang et al. (Tang et al., 2007) defined assumptions as "explicit documentation of the unknowns or the expectations to provide a context to decision making", which is an important architecture element in rationale-based architecture model.

Lewis et al. (Lewis et al., 2004) proposed a prototype Assumptions Management System in software development, which can extract assumptions from source code and record them into a repository for management. The authors also provided a taxonomy of general assumptions in software development, including (1) control assumptions (expected control flow), (2) environment assumptions (expected environmental factors), (3) data assumptions (expected input or output data), (4) usage assumptions (expected use of applications), and (5) convention assumptions (followed standards or conventions in development).

Tirumala et al., (2005) focused on component assumptions, and emphasized that mismatched assumptions between software components are one of the major reasons leading to failures in real-time systems, and most component assumptions are implicit. Thus the authors proposed a framework to make component assumptions explicit in real-time systems.

Zschaler and Rashid, (2011) focused on aspect assumptions in aspect-oriented software development, and classified aspect assumptions in two categories, with six types and thirteen subtypes. This classification is beneficial for code improvement, and assumptions elicitation and verification in aspect-oriented code.

Haley et al., (2006) focused on trust assumptions, and pointed out that trust assumptions (including explicit and implicit trust assumptions) may impact the way to realize functions of a system and the scope of requirements analysis. The authors also proposed a model, which is composed of six elements, to present trust assumptions.

Lehman and Ramil, (2001) proposed several guidelines for managing assumptions. For example, the authors believed that it is necessary to train all stakeholders to identify and record assumptions (including explicit and implicit assumptions) at all stages of the development based on a standard form or structure.

These works investigate several types of assumptions (e.g., component assumption, aspect assumption, and trust assumption), which focus on different aspects of a system, while our survey focuses on architectural assumption. The related work reveals a number of interesting findings and directions on assumptions in software engineering (e.g., definitions of assumptions, classifications of assumptions, and approaches of identifying and recording assumptions), which have been used as input for this survey with a special focus on AA (see Section 3.3.1). In addition, in this survey we deal with the use of the AA term, the importance of AA, as well as the challenges (reasons) of (not) identifying and recording AA.

2.2. Architectural assumptions

Garlan et al. (Garlan et al., 2009) identified four general categories of AA that are implicit and undocumented and consequently lead to architectural mismatch: (1) nature of components, (2) nature of connectors, (3) global architectural structure, and (4) software construction process. This categorization is based on a structural view of architecture, which regards SA as a set of structures, including components and connectors.

Lago and van Vliet, (2005) distinguish AA from requirements and constraints as the reasons for ADDs 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 groups: (1) managerial assumptions, (2) organizational assumptions, and (3) technical assumptions.

Ostacchini and Wermelinger, (2009) 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 assumptions proposed by Lago and van Vliet, (2005).

Roeller et al., (2006) classified AA into four groups as: (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 developed a method (Recovering Architectural Assumptions Method, RAAM) to recover assumptions in architecture design. Furthermore, AA were defined as a type of ADDs as well as the reasons for making the decisions, and the AA term was used as a general denominator for the forces that drive ADDs.

Yang and Liang, (2014), we proposed a lightweight approach for identifying and recording AA in agile development, which is composed of a simplified AA conceptual model and a process to identify and record AA. For this work, we used the same definition of AA used by Roeller et al., (2006).

Van Landuyt et al., (2012) focused on early AA in scenario-based requirements, and defined AA as “typically assumptions about the structure of the system under development, i.e., the existence of particular system elements (e.g. components)”, which is different from the definition of Lago and van Vliet, (2005). Furthermore, the authors highlighted the need to model AA explicitly and precisely in the early phases of software development (i.e., requirements engineering and the transition to architecture). In their follow-up work, Van Landuyt and Joosen, (2014) proposed a set of specific techniques (e.g., a system meta-model and an aspect-oriented requirements language) to modularize early AA in the context of scenario-based requirements.

The aforementioned work focuses on several aspects of AA (e.g., classification of AA and approaches for AA management) using e.g., case studies and interviews. To the best of our knowledge, there is no industrial survey on the topic of AA, and our survey intends to provide an overview of the understanding of AA, as well as the identification and recording of AA from the practitioners' perspective.

3. Research approach

According to the objectives of this study as well as the guidelines for selecting empirical methods in software engineering research proposed by Easterbrook et al., (2008), we decided to use a survey to identify the state of art of AA in industry. This is because we need to identify the characteristics of a broad population (software architects and other software professionals) regarding how they understand, identify, and record AA. We do not aim at exploring what happens when practitioners manage AA (in which case a case study can be employed), or study correlation or causality of variables related to AA (in which case an experiment is more appropriate).

A survey is not just the instrument (the questionnaire or checklist) to gather information (Kitchenham and Pfleeger, 2008), it is a comprehensive research method for describing, comparing, or explaining knowledge, attitudes, and behavior (Fink, 1995). We followed and adapted the guidelines proposed by Kitchenham and

Pfleeger, (2008) to conduct the survey and designed the survey protocol according to a template for survey protocols in evidence-based software engineering². In the rest of this section we report the survey design partially based on the protocol. The research questions (RQs) are defined in Section 3.1. The form of the survey, the sample and the population, the selection criteria, and the sampling methods we used are described in Section 3.2. The preparation of the questionnaire and the collection of the responses are elaborated in Section 3.3. The data analysis of the survey results is presented in Section 3.4, and the survey instrument is evaluated in Section 3.5.

3.1. Research questions

We formulated the RQs in Table 1 based on the survey objectives presented in Section 1.

3.2. Survey design

3.2.1. Form of the survey

The survey is descriptive (describing the characteristics of a population), and the survey design is a cross sectional study (collecting the data at a fixed point in time) (Kitchenham and Pfleeger, 2008). An online questionnaire was used as a data collection instrument, because web-based questionnaires are time- and cost-effective (Lethbridge, 2005). Questionnaires are also well-suited to collect data from a large number of persons in geographically diverse locations (Lethbridge, 2005).

3.2.2. Sample and population

The target population was limited to the practitioners who use Chinese as native language, work in China, and have experience in software architecture design or are involved with software architecture (e.g., project managers). The reason for selecting Chinese-speaking respondents is that we used Non-Probabilistic Sampling method in this survey (see Section 3.2.4) and Chinese respondents are easily accessible to two of the researchers of this survey. The population of software engineers in China is one of the biggest in the world, so it makes good sense to sample from this population. We asked several questions to determine their role and background, such as the main tasks they perform, number of years working in IT industry, and number of years of experience in software architecting.

We used three approaches to reach the target population:

- (1) The contacts in our social networks (such as previous coworkers).
- (2) Websites on software development. Three popular technical websites³ (in Chinese) were used to find and invite potential participants of this survey, who have the title of architect or show an interest in SA (e.g., s/he posted a blog on SA).
- (3) Professional or academic conferences or symposiums. One author attended a professional architects' conference (International Architect Summit organized by InfoQ⁴) in Beijing and personally invited a number of participants.

3.2.3. Obtaining a valid sample

The inclusion and exclusion criteria in Table 2 were defined to select valid responses for analysis. Boolean OR is used to connect alternate criteria in both the inclusion and exclusion criteria. The authors discussed and reached a consensus on the understanding

² <http://community.dur.ac.uk/ebse/resources/templates/SurveyTemplate.pdf>.

³ <http://www.cnblogs.com/>, <http://www.csdn.net/>, and <http://s.weibo.com/>.

⁴ <http://bj2014.archsummit.com/>.

Table 1
Research questions and their rationale.

Research question	Rationale
RQ1: What is the practitioners' perception of AA and importance of AA in software development?	This RQ helps to study how the AA term is used and defined, what examples of AA practitioners have in mind and how important it is to manage AA during architecting and software development.
RQ2: What are the approaches, tools, as well as the challenges in identifying and recording AA? RQ2.1: What are the approaches used for identifying and recording AA? RQ2.2: What are the tools used for identifying and recording AA? RQ2.3: What are the challenges in identifying and recording AA?	This RQ sheds light on what approaches and tools are actually used in practice for identifying and recording AA, and helps researchers to identify possible improvements to the approaches and tools.
RQ3: What are the major reasons for not identifying and recording AA?	This RQ helps to identify the major impediments in identifying and recording AA in software development.

Table 2
Inclusion and exclusion criteria for selecting responses in the survey.

Inclusion criteria
I1: The respondent has experience in software architecting.
I2: The respondent has no experience in software architecting, but has experience in software design.
Exclusion criteria
E1: Inconsistencies exist in the response.
E2: The response is meaningless.
E3: The respondent does not work in China.

of these criteria before selecting the responses. Note that the inclusion criterion I1 was evaluated by the answer of survey question DQ5 (we asked the respondents their experience (in years) in software architecting), I2 was evaluated by the answer of survey question DQ3 (we asked the respondents their main tasks in their companies), and E3 was evaluated by the answer of survey question DQ1 (we asked the respondents the country they are working in). The two other exclusion criteria (i.e., E1 and E2) are detailed below.

"Inconsistencies" in the exclusion criterion E1 means two or more answers in a response are conflicting. For example, a participant chose "Yes" to the question "SQ1: Are you familiar with the term Architectural Assumption?" but answered "No idea" to the question "SQ5: Can you provide an example of architectural assumption (or examples if you think there are more than one types) according to your understanding?"

"Meaningless" in the exclusion criterion E2 means that what a participant wrote is gibberish or logically senseless to some or all the open questions.

3.2.4. Sampling methods

The Non-Probabilistic Sampling method (including Convenience Sampling and Snowball Sampling) was employed to obtain a valid sample, because Chinese-speaking respondents are easily accessible to two of the researchers of this survey (mentioned in Section 3.2.2). Convenience Sampling refers to acquiring responses from available people who are willing to participate (Kitchenham and Pfleeger, 2008). Convenience Sampling was used in this study because several sources (e.g., websites on software development) are available to invite the participants. Snowball sampling means asking the respondents in a survey to invite other available people they consider would be willing to participate (Kitchenham and Pfleeger, 2008).

3.3. Data preparation and collection

3.3.1. Creating the questionnaire

Through reviewing the published literature on assumptions (see Section 2), a survey instrument consisting of nine questions on demographics (DQ1–DQ9) and fifteen specific questions (SQ1–SQ15) was developed (see Appendix). Questions on demographics are used to collect personal information (e.g., the country and the main tasks) and company information (e.g., the size of the company and the domain of the company) to help the authors better understand the background of the participants. Specific questions are used to collect the information to answer the research questions stated in Section 3.1 (e.g., the definition of AA and the example of AA).

Roeller et al., (2006) stated that different stakeholders may have different understanding of the AA concept from different perspectives, and Lago and van Vliet, (2005) pointed out that "it is quite tricky to draw the line between assumptions, requirements and constraints". To understand how far practitioners are familiar with the term of AA and how they perceive AA, we designed five survey questions: SQ1–SQ5. The related work discussed in Section 2 emphasizes that assumptions (e.g., AA) are important in software development (e.g., architecting) (Roeller et al., 2006) and problems (e.g., architectural mismatch) can be caused by vaporization of assumptions (e.g., implicit assumptions) (Garlan et al., 2009). Based on this, we designed two survey questions, SQ6 and SQ7, in order to understand how practitioners consider the importance of AA in their work. SQ8 and SQ9 are used to explore how practitioners identify AA, which is a concern raised in e.g., (Ostacchini and Wermelinger, 2009) and (Yang and Liang, 2014). SQ12 and SQ13 were designed to investigate how AA are recorded in practice, which is a concern discussed in e.g., (Lago and van Vliet, 2005) and (Yang and Liang, 2014). Since AA vaporization will negatively impact AA management, it is important to explore the challenges of identifying and recording AA as well as the reasons for not identifying and recording AA. To this end, we designed questions SQ10, SQ11, SQ14, and SQ15. As shown in Fig. 1, specific questions are further classified into five parts, and each part is related to one or more RQ it aims to address.

As shown in Appendix, questions DQ1, DQ4, DQ5, DQ9, SQ3, SQ4, SQ5, SQ9, and SQ13 are open questions (free text), questions DQ6, DQ7, SQ1, SQ2, SQ6, SQ7, SQ8, and SQ12 are closed questions (single or multiple choice from a list), while DQ2, DQ3, DQ8, SQ10, SQ11, SQ14, and SQ15 are semi-closed questions (Del Greco and Walop, 1987) (i.e., choice from a list with the option to add free text).

Fig. 1. Questionnaire flow.

3.3.2. Questionnaire format

Every question was placed in a different web page. We set a welcome page to explain the purpose of the survey, with an everyday example of assumption, and our affiliation and contact information, etc., which is partially shown in [Appendix](#).

For the implementation of the survey, an online web-based survey tool LeDiaoCha⁵ (in Chinese) was used. Participants visited the survey hosted at LeDiaoCha through a web link. The survey questions are also shown in [Appendix](#).

3.4. Data analysis

For data analysis we used descriptive statistics for quantitative answers (to multiple choice questions, such as the educational background of the respondents), and used Grounded Theory (detailed in the next paragraph) for qualitative answers (where respondents answered an open question).

Grounded Theory was used to generate categories from the responses to answer the RQs defined in [Section 3.1](#). In this survey, the categories are the classification of the ways of using the AA term, definitions of AA, examples of AA which are presented in [Section 4.2](#), and the methods of identifying and recording AA which are described in [Sections 4.4](#) and [4.5](#) respectively. The three coding phases defined in classical Grounded Theory are open coding, selective coding, and theoretical coding ([Adolph et al., 2011](#)). Note that only open coding and selective coding were used, because we only needed to acquire categories of the investigated subjects (e.g., AA definition) instead of generating theories. Open coding generates codes that can be clustered into concepts (breaking data into conceptual components) and categories (aggregating a set of concepts) ([Corbin and Strauss, 2014](#)). This phase was used to generate codes of certain data items (e.g., definitions of AA presented in [Section 4.2](#)). Selective coding identifies the core category (e.g., the five types of AA definitions), which explains the greatest variation in the data ([Corbin and Strauss, 2014](#)). Grounded Theory was executed in an iterative process with two of the authors, and the codes with their relationships were refined and adapted in each iteration. The third author was consulted to resolve inconsistencies and review the results for clarity.

Furthermore, if an answer provided by a respondent leads to more than one possible interpretation, we regarded the answer as invalid, and removed that answer, but still kept the response. Note that in this survey, an “answer” is related to one survey question,

and a “response” is composed of all the valid answers of the questionnaire by a respondent.

3.5. Survey instrument evaluation

The protocol of this survey was refined iteratively by the three authors and also externally reviewed by two additional researchers. The questionnaire was reviewed externally by two other additional researchers as well as three practitioners on software engineering and architecture, for checking the content and the understandability of the questions, as well as the approximate time needed to complete the questionnaire. Furthermore, a pilot study was run with seven participants, and we interviewed three of them to further refine the survey instrument (e.g., adding an example of assumption from everyday life). The data points from the pilot were included in the survey as we did not change the survey questions after the pilot study. The participants of the pilot study are representative of the potential respondents of this survey (covering various years of experience, i.e., less than 5, between 5 and 10, and more than 10 years in IT industry).

4. Survey results

This section presents the survey results based on the collected data, which follows the structure of the questionnaire described in [Section 3.3.1](#). The five subsections (from [Sections 4.1–4.5](#)) correspond to the six parts of the questionnaire shown in [Fig. 1](#) (i.e., *d* and *s1* to *s5*).

4.1. Demographic data

In total 213 responses were received of which 101 (47.4%) invalid responses were excluded based on the selection criteria (see [Section 3.2.3](#)); the remaining 112 valid responses were further analyzed. Note that, the term “respondents” as used in the rest of the paper refers to the respondents who provided valid responses to this survey.

The respondents of this survey are located in 23 cities of China. The top 5 cities of respondents are listed in [Table 3](#). The most popular cities are Beijing (24 out of 112, 21.4%) and Shenzhen (21 out of 112, 18.8%), which are also at the top of the list of the IT industry in China.

The respondents were classified according to their educational background (B.Sc., M.Sc., Ph.D., and Others), as shown in [Fig. 2](#). Most respondents (69 out of 112, 61.6%) hold a B.Sc. degree, while 31 (27.7%) respondents have an M.Sc. degree.

⁵ <http://www.lediaochoa.com/>.

Fig. 2. Educational background and main tasks of respondents.

Fig. 3. Number of years of experience in IT industry and in software architecting of the respondents.

Table 3
Top 5 Cities of respondents.

City	Frequency	Percent (%)
Beijing	24	21.4
Shenzhen	21	18.8
Shanghai	11	9.8
Guangzhou	9	8.0
Wuhan	8	7.1

We asked the respondents about their main tasks in their companies, so that we better understand their role. Note that respondents could select one or more main tasks. As shown in Fig. 2, most respondents selected “Development” (72 out of 112, 64.3%) as their main task, and “Architecture design” and “Detailed design” follow behind with 61 (54.5%) and 56 (50.0%) respondents respectively, which shows that development and design are the main tasks of the respondents. Additionally, one respondent chose the option “Others” and provided the answer “Operation and maintenance” as the main task.

The respondents were further classified according to their working experience in IT industry and software architecture into three types as shown in Fig. 3(a). Most respondents (72 out of 112, 64.3%) have at least 5 years of working experience in IT industry. Fig. 3(b) shows that most respondents (70 out of 112, 62.5%) have less than 5 years of experience in software architecting.

We also asked whether the respondents received professional training on software architecture or design. We explicitly mentioned in DQ6 (see Appendix) that professional training does not include higher education (e.g., software architecture course at university), because we wanted to distinguish between industrial training with a hands-on assignment tied to the daily practice of architects and working on purely academic course projects. Only 25.0% (28 out of 112) respondents answered that they had such training (see the left part of Fig. 4).

The size of the companies where the respondents work was classified into four types as shown in the right part of Fig. 4. Over half of the respondents (58 out of 112, 51.8%) worked in companies with more than 250 employees, and only 3 (2.7%) respondents worked in small companies (with less than 10 employees).

Fig. 4. Professional training related to software architecture or design of the respondents (left) and size (in persons) of the companies where the respondents work (right).

Fig. 5. Top 6 domains of the companies where the respondents work (left) and development methods used in the projects that the respondents were involved (right).

The left part of Fig. 5 shows the top 6 domains of the companies in which the respondents work. “IT services” (69 out of 112, 61.6%), which includes the services as well as the process of providing the services on information technology, is the most popular domain, far ahead from the others. “Telecommunications” (25 out of 112, 22.3%), “E-commerce” (23 out of 112, 20.5%), and “Embedded systems” (23 out of 112, 20.5%) are in the second, third, and fourth place respectively. 23.2% (26 out of 112) of the respondents chose the option “Others” and provided specific domains of their companies. They are “Retail” (3), “Insurance” (3), “Social” (2), “Game” (2), “Internet” (2), “Real estate” (1), “Electrical power system” (1), “Smart home” (1), “Enterprise resource planning system” (1), “Energy” (1), “Construction” (1), “Education” (1), “Transportation” (1), “Printing” (1), “Government” (1), “Security system” (1), “Media” (1), “Logistics” (1), and “Manufacturing” (1).

The right part of Fig. 5 shows the development methods commonly used in the projects that the respondents were involved. The related survey question DQ9 (see Appendix) is an open

question. Most respondents (71 out of 112, 63.4%) used “Iterative development”, while “Waterfall model” (34 out of 112, 30.4%) and “Agile development” (24 out of 112, 21.4%) are ranked in the second and third position respectively. It is worth noting that there is potentially an overlap between “Iterative development” and “Agile development” methods: agile methods are iterative, and are considered as a subset of iterative methods (Larman, 2004; Larman and Basili, 2003). Since this is an open question and we just collected the answers that the respondents provided, we urge readers to interpret them with caution, considering a potential overlap.

4.2. Understanding of architectural assumption

As little has been known about how AA is understood and how important AA is considered by practitioners, a set of survey questions (SQ1–SQ7 in Appendix) were used to explore these aspects. We first asked the respondents whether they are familiar with the AA term (SQ1 in Appendix) and whether they used the term in

Fig. 6. Familiarity and use of the architectural assumption term.

Table 4
Definition of architectural assumption.

Type	Definition of architectural assumption	Count
Requirement	AA are a type of uncertain requirements (functional or non-functional).	13
Architecture	AA represent conjectures about the architecture (e.g., assumptions about scenarios to support architecture evaluation).	14
Constraint	AA are regarded as constraints on a system or constraints on an architecture.	14
Risk	AA are considered as inputs to manage risks (e.g., for risk prediction).	1
Project context	AA are conjectures about the project context.	4

their projects (SQ2 in [Appendix](#)). The results in [Fig. 6\(a\)](#) and [\(b\)](#) show that most respondents are not familiar with the term (75 out of 112, 67.0%), and never used this term in their work (98 out of 112, 87.5%).

The respondents were asked to define AA according to their own knowledge and experience in an open question (SQ4 in [Appendix](#)). This was an optional question and 61 respondents provided an answer. We collected 46 (46 out of 61, 75.4%) valid answers and classified the AA definitions into five types as listed in [Table 4](#).

In addition to the definitions of AA, we also used an open question (SQ5 in [Appendix](#)) to ask the respondents to provide examples of AA according to their experience, so that the provided definitions can be exemplified. We classified the AA examples into three types, which are a subset of the types of the definitions of AA ([Table 4](#)), with representative examples listed in [Table 5](#).

- (1) “Requirement”: Use AA as requirements with certain level of uncertainty. This type is further classified into two subtypes: “Functional requirement” and “Non-functional requirement”.
- (2) “Architecture”: Use AA with regard to architecting, or its technical environment.
- (3) “Project context” type is classified into two subtypes: “Organization” and “Business”. “Organization” type concerns the company’s structure and the profile of the development team ([Roeller et al., 2006](#)), while “Business” type concerns the business objectives of a project.

If the respondents did use the AA term, we asked how they used it through an open question (SQ3 in [Appendix](#)). We classified the ways of using the AA term into five types, and we list both the types and the answers in [Table 6](#). These are the same types that were used to classify the definitions of the AA term, as presented in [Table 4](#). Using the AA term to represent uncertain requirements (including functional and non-functional requirements) is the most common way of using this term.

4.3. Importance of architectural assumptions

The respondents were asked to rank the importance of AA in software architecting (SQ6 in [Appendix](#)) as well as in the entire software development lifecycle (SQ7 in [Appendix](#)). Note that this survey concerns the importance of the AA concept, not the AA term. As shown in [Fig. 7](#), over half of the respondents considered that AA are important in both software architecting (51.8%, 58 out of 112) and the software development lifecycle (50.9%, 57 out of 112); note that more respondents considered that AA are “very important” in software architecting than in the software development lifecycle (37 vs. 24). About one fifth of the respondents stated that they had no idea about the importance of AA.

4.4. Identifying architectural assumptions

With regard to AA identification, we paid special attention to how the respondents identified AA in their projects, the challenges of identifying AA, and the reasons for not identifying AA. [Fig. 8](#) shows that more than half of the respondents (56.3%, 63 out of 112) had never identified AA in their projects.

It is important to explore which type of support has been provided for practitioners to facilitate AA identification. We asked the respondents who identified AA in their projects, how they identified them (before or after the fact) through an open question (SQ9 in [Appendix](#)). [Table 7](#) shows the approaches of AA identification collected from the responses, which are classified into six types:

- (1) “During development process”: AA were identified during one of the phases of the development process (e.g., during requirements engineering).
- (2) “In documents”: AA were identified from the documents of projects (e.g., in architecture documents).
- (3) “Through communication”: AA were identified through discussion between stakeholders.

Table 5
Example of architectural assumption.

Type	Subtype	Example of architectural assumption
Requirement	Functional requirement	Two representative examples: <ul style="list-style-type: none"> • The architect assumes that the functions of the system depend on the specific domain and the users. • The architect assumes that the system should provide the function of automatic error correction.
	Non-functional requirement	Four representative examples: <ul style="list-style-type: none"> • The architect assumes that the response time of the system should be less than 0.1 s. • The architect assumes that the number of concurrent users of the system would be around 200. • The architect assumes that the system needs to be easily extended (i.e., extensibility). • The architect assumes that the customers may use both Microsoft SQL Server database and Oracle database in this project.
Architecture		Four representative examples: <ul style="list-style-type: none"> • The architect assumes that concurrency, integration, and interfaces of the system should be the three main factors for selecting the architecture. • The architect assumes that the application framework can be applied (reused and extended) in most projects in the company. • The architect assumes that the external distributed file system and content delivery network can be used in the project. • The architect assumes that the architecture design of the system needs to be easily reused and decomposed.
Project context	Organization	Three representative examples: <ul style="list-style-type: none"> • The architect assumes that the skills and capacities of the software engineers in the development team are enough for this project. • The architect assumes that the development team can understand 60% of the requirements from users. • The architect assumes that the development team should have a clear understanding of the field of mobile networks.
	Business	Four representative examples: <ul style="list-style-type: none"> • The architect assumes that the project should be completed in one month. • The architect assumes that the potential users of the system should be from all over the world. • The architect assumes that the image recognition component from the third party is available for this project. • The architect assumes that 90% of the business processes implemented in the project can be standardized.

Table 6
Ways of using the architectural assumption term.

Type	Ways of using the architectural assumption term
Requirement	Use AA to represent uncertain non-functional requirements. Use AA to represent uncertain functional requirements.
Architecture	Use AA to make assumptions explicit and clear to stakeholders during architecture design.
Constraint	Use AA to assume system constraints under uncertain situations.
Risk	Use AA as inputs for risk management.
Project context	Make AA about the context of project (e.g., system size) explicit, in order to elaborate the business model, and estimate the cost and lifecycle of the project.

- (4) “Experience and knowledge”: AA were identified based on personal experience and knowledge.
- (5) “No specific approach” indicates that the respondents identified AA in their projects in an ad-hoc manner, without using a specific source or approach.
- (6) “Others” contains the approaches, which cannot be classified into the five types above.

49 respondents had identified AA in their projects as shown in Fig. 8. Most respondents identified AA during the requirements engineering process (30.6%, 15 out of 49), or based on their personal experience and knowledge (24.5%, 12 out of 49). Additionally, 14.3%

(7 out of 49) respondents stated that they identified AA in testing, and 8.2% (4 out of 49) respondents identified AA without using any specific source or approach.

Only three tools (Rational Rose, Zentao⁶, and MS Excel) have been used for AA identification by the respondents (each used by one respondent), and none of them are dedicated to identifying AA. Note that respondents could provide both the approaches and tools they used for AA identification, or only one of them.

⁶ <http://www.zentao.net/> (a project management tool, available in Chinese).

Fig. 7. Importance of architectural assumption.

Fig. 8. Identifying architectural assumptions in software development.

Next, it is important to identify the challenges that hinder the identification of AA, so we posed a semi-closed question to explore this aspect. The respondents were given a list of candidate challenges on AA identification according to our experience and knowledge (e.g., “Lack of approaches” and “Lack of tools”) to choose from. The respondents could also provide other challenges that they thought important for identifying AA.

Note that the candidate challenges (1) “Lack of tools”, “Lack of approaches”, and “Lack of guidance” denote the tools, approaches, and the guidance used for identifying AA, (2) “Lack of experts” refers to experts who are familiar with the AA concept and its management, and (3) “Unawareness” means that developers make assumptions implicitly without being aware of them. In the questionnaire, the respondents could read the full description of these

challenge options. As shown in Fig. 9, “Lack of tools” (25 out of 49, 51.0%) is the most important challenge in AA identification. “Lack of experts” (23 out of 49, 46.9%) and “Lack of approaches” (21 out of 49, 42.9%) follow behind. Five respondents provided five other challenges of identifying AA as listed below.

- (1) Lack of support from customers (2 out of 49, 4.1%)
- (2) Lack of experts of specific domains (1 out of 49, 2.0%)
- (3) Project requirements are always vague (1 out of 49, 2.0%)
- (4) Lack of resources to support AA identification (1 out of 49, 2.0%)
- (5) Lack of support from the development team (1 out of 49, 2.0%)

For the respondents who never identified AA in their projects, we asked them to motivate their answers in a semi-closed question. The respondents were given a list of potential reasons for not identifying AA (e.g., “Lack of approaches” and “Lack of time”) to choose from according to their experience. The respondents could also provide other reasons that they thought were important for not identifying AA.

As shown in Fig. 10, “Lack of approaches” (30 out of 63, 47.6%) is ranked in the first position. “No time” (16 out of 63, 25.4%), “Lack of tools” (15 out of 63, 23.8%), and “Costs outweigh benefits” (15 out of 63, 23.8%) follow behind. Thirteen respondents provided three other reasons for not identifying AA as listed below.

- (1) Unfamiliar with AA either as a term or as a concept (9 out of 63, 14.3%)
- (2) Making assumptions implicitly without being aware of them (3 out of 63, 4.8%)
- (3) When everything in the software project can be determined, it is not necessary to make any AA, and consequently no AA should be identified (1 out of 63, 1.6%)

4.5. Recording architectural assumptions

With regard to AA recording, we focus on how the respondents recorded AA in their projects, the challenges of recording AA,

Fig. 9. Challenges of identifying architectural assumptions.

Table 7
Approaches of identifying architectural assumptions.

Type	Approach	Count
During development process	In the requirements engineering process (e.g., in requirements analysis)	15
	When testing the system	7
	When implementing the system	1
In documents	In architecture design documents	2
	In project plan documents	1
Through communication	Through the discussion within the development team (including formal and informal meetings)	3
	Through the communication with customers (e.g., in each iteration)	2
	Through the feedback from users	2
Experience and knowledge	Based on the personal experience and knowledge of architects	12
	Based on the personal experience and knowledge of domain experts	2
No specific approach	Without using any specific approach	4
Others	Through analyzing the existing data (e.g., transaction data) of the system	2
	Through exploring uncertain factors, which may have an impact on architecture	1
	Through decision tables analysis (Vanthienen and Wets, 1994)	1

and the reasons for not recording AA. Fig. 11 shows that most respondents (80.4%, 90 out of 112) had never recorded AA in their projects.

It is important to know which type of support has been provided for practitioners to record AA. Considering this, for the respondents who recorded AA in their projects, we asked them how they recorded AA through an open question (SQ13 in Appendix). As shown in Table 8, only six different tools were used and none of them are dedicated to recording AA. Similar to the AA identification question (SQ9), respondents may provide both the approaches and tools they used for AA recording, or just one of them.

22 respondents had recorded AA in their projects as shown in Fig. 11. However, only two approaches for AA recording were identified, as shown in Table 9. Most respondents (16 out of 22, 72.7%) recorded AA in various development documents.

Similar to the question (SQ10) about the challenges of identifying AA, we asked a semi-closed question (SQ14) in order to explore the challenges of AA recording. The respondents could also provide other challenges that they thought important for recording

AA. The results are shown in Fig. 12. In summary, “Lack of tools” (8 out of 22, 36.4%) and “Lack of management support” (8 out of 22, 36.4%) are the most important challenges of AA recording. One respondent provided one other challenge of recording AA, as “Project requirements are always vague.”

Similar to the reasons for not identifying AA, we asked a semi-closed question (SQ15) to explore the reasons for not recording AA. The respondents could also provide other reasons for not recording AA according to their experience. As shown in Fig. 13, “Lack of approaches” (46 out of 90, 51.1%) is ranked in the first position. “Lack of tools” (35 out of 90, 38.9%) and “No time” (24 out of 90, 26.7%) follow behind. Fifteen respondents provided three other reasons for not recording AA as listed below.

- (1) Unfamiliar with AA either as a term or as a concept (9 out of 90, 10.0%)
- (2) Making assumptions implicitly without being aware of them (5 out of 90, 5.6%)
- (3) AA are not considered an important issue (1 out of 90, 1.1%)

Fig. 10. Reasons for not identifying architectural assumptions.

Fig. 11. Recording architectural assumptions in software development.

Table 8

Tools of recording architectural assumptions.

Tool	Count
MS Excel	3
SVN	1
Wiki	1
OneNote	1
Evernote	1
Internal tools used in the company	1

Table 9

Approaches of recording architectural assumptions.

Approach	Count
Record AA in different documents (e.g., requirements documents, design documents, test documents, and meeting notes)	16
Record and embed AA when implementing the system in code (both source code and code comments).	1

Fig. 12. Challenges of recording architectural assumptions.

Fig. 13. Reasons for not recording architectural assumptions.

5. Discussion of results

The results indicate that AA are important in both software architecting and the whole software development lifecycle (see Fig. 7). However, identifying and recording AA is not yet a common practice: 49 respondents identified AA (in Fig. 8), but only 22 respondents recorded AA in their projects (in Fig. 11). This indicates that AA are kept implicit and undocumented in most cases. Furthermore, unfamiliarity of AA as a concept, and the lack of spe-

cific approaches and tools are the major challenges of identifying and recording AA; at the same time these are also the main reasons for not identifying and recording AA at all. Several interesting findings are discussed in detail below.

5.1. Understanding of architectural assumption

Familiarity and use of AA term and concept: Note that “the AA term” is not the same as “the AA concept”. A term is a label

Table 10
Comparison of architectural assumption definitions.

Source	Architectural assumption definition
Our survey	Requirement: AA are a type of uncertain requirements (functional or non-functional). Architecture: AA represent conjectures about the architecture (e.g., assumptions about scenarios to support architecture evaluation). Constraint: AA are regarded as constraints on a system or constraints on an architecture. Risk: AA are considered as inputs to manage risks (e.g., for risk prediction). Project context: AA are conjectures about the project context.
Lago and van Vliet (2005)	The reasons for ADDs that are arbitrarily taken based on personal experience and knowledge.
Roeller et al. (2006)	A type of ADDs as well as the reasons for making the ADDs.
Van Landuyt et al. (2012)	Typically assumptions about the structure of the system under development (e.g., components).

used to express a concept (i.e., semantics) through a word or phrase (i.e., syntax), while a concept conveys the meaning of a term. As shown in Fig. 6, only 33.0% (37 out of 112) respondents were familiar with the AA term and only 12.5% (14 out of 112) respondents used the term in their projects. One interpretation may be that there is no popular and consistent translation of the AA term into Chinese. Respondents may know what the AA concept means, but use different terms to describe the AA concept. In the pilot study, we made three interviews. One objective of the interviews was to find a term of AA in Chinese commonly used by practitioners. However, no participants of the pilot study could recommend a better term than the AA term used in the pilot survey although they conjectured that the one used is not perfect. They all confirmed that the AA concept is used in software development in China. However, there is no widely accepted AA term in Chinese translation.

Though 67.0% respondents (75 out of 112) were not familiar with AA as a term and 87.5% respondents (98 out of 112) had never used the term in their work, the results from other questions (such as SQ4 and SQ5) showed that most of them understood the concept of AA. This is evident in Tables 4 and 5, where we collected some meaningful definitions and examples of AA from the responses. It is a typical issue in Software Engineering that terms are not commonly defined and agreed upon. Many terms are intuitively understood by practitioners without an agreement on their exact meaning.

AA definition: Over half (54.5%, 61 out of 112) of the respondents answered the open and optional question (SQ4) about AA definition. As shown in Table 4, five types of AA definitions were collected and identified, which indicates that respondents had different understanding on the AA concept (Lago and van Vliet, 2005). Most respondents considered AA as a type of uncertain requirements, conjectures about the architecture, or constraints on a system or an architecture. Note that requirement assumptions in (Haley et al., 2006) and the “Requirement” type of AA identified through this survey are similar. The relationship between these two types of assumptions is that all the “Requirement” type of AA are uncertain architecturally significant requirements (e.g., the architect assumes that the number of users (visitors) of the system would be around 1 million per day), which are a subset of requirement assumptions. A comparison of the AA definitions got in this survey with existing AA definitions is provided in Table 10, and the results of this survey show that AA are related to various aspects of a system, but not limited to architectural elements (ADDs or system structure). This is in line with the findings presented in Fig. 7: AA is regarded as an important element, in both architecting and the whole software development lifecycle, and its man-

agement should be spread throughout the software development lifecycle.

Considering the various definitions of AA in this survey (as shown in Table 4), we argue that AA identification and recording should not be limited to a single software development phase (e.g., architecting), but managed throughout the software development lifecycle. Furthermore, prior to applying AA identification and recording in a project, reaching a common understanding of AA among different stakeholders in the project is needed and may be critical to the successful utilization of AA.

AA example: 73.2% (82 out of 112) respondents gave at least one example of AA (question SQ5). It was surprising that even the respondents, who had never identified or recorded AA in their projects, still came up with meaningful AA examples. According to the responses, we developed a new classification of AA with three types and four subtypes as shown in Table 5, which is a subset of the five types of AA definitions in Table 4. The AA classification developed in this survey based on AA examples is partially overlapping with existing AA classifications as shown in Table 11. For example, nature of components and nature of connectors (Garlan et al., 2009) can be mapped to “Architecture”, and managerial assumptions and organizational assumptions (Lago and van Vliet, 2005) can be mapped to “Organization” (in Table 5). However, neither of the classifications ([5] and (Lago and van Vliet, 2005)) included “Requirement” (which in our case is architecturally significant requirement) and “Business” (e.g., assumptions about project deadlines) as AA types. This finding indicates that AA is understood in different ways by various stakeholders depending on their focus and interests (e.g., business, requirements, or architecture).

AA importance: As shown in Fig. 7, more than 50% of the respondents considered that AA are important in both software architecting (51.8%, 58 out of 112) and the software development lifecycle (50.9%, 57 out of 112). This result indicates that it is worth to manage AA (e.g., identifying and recording AA) in projects. Furthermore, 33.0% (37 out of 112) of the respondents thought AA are “very important” in software architecting, but only 21.4% (24 out of 112) considered AA are “very important” in the whole software development lifecycle. This is not a surprising result, because AA as well as its management are mainly related to architecting.

5.2. Challenges and impediments

The survey results (see Figs. 9, 10, 12, and 13) show that the challenges of identifying and recording AA and the reasons for not identifying and recording AA (impediments) are largely overlapping. In summary, we classified them into four types:

Table 11

Comparison of classifications of architectural assumption examples.

Source	Classification of architectural assumption examples
Our survey	Requirement (Functional requirement and Non-functional requirement) Architecture Project context (Organization and Business)
Garlan et al. (2009)	Assumption about the nature of components Assumption about the nature of connectors Assumption about the global architectural structure Assumption about the construction process (development environment and build)
Lago and van Vliet (2005)	Managerial assumption Organizational assumption Technical assumption
Roeller et al. (2006)	Implicit and undocumented assumption Explicit but undocumented assumption Explicit and explicitly undocumented assumption Explicit and documented assumption

- (1) Lack of approaches, tools, and guidelines for AA identification and recording (mentioned as both challenges and impediments)

This finding has also been discussed in [Section 5.3](#) regarding the lack of approaches and tools for AA identification and recording, so we do not repeat it here.

- (2) Time constraint of projects (mentioned as both challenges and impediments)

Industrial projects are by and large under time pressure; this is also the case for industry in China. Several factors such as market pressure of fast delivery or project deadlines force development teams to ignore AA identification and recording. One of the respondents interviewed in the pilot study said: *“It may be risky, if the development team does not identify and record AA. However, it can save time, which is more important to the whole project.”* Providing lightweight approaches with supporting tools for AA identification and recording may partially alleviate this problem.

- (3) Characteristics of respondents (mentioned as both challenges and impediments)

Characteristics of the respondents include e.g., their experience, knowledge, and personal preferences. The lack of a common understanding of the AA concept was regarded as an important impediment for not identifying and recording AA. The respondents emphasized that if they are not familiar with the AA concept, they could neither identify nor record AA in projects, which is also emphasized by [Roeller et al., \(2006\)](#). As shown in [Figs. 8 and 11](#), only a few respondents identified and recorded AA in their projects. One challenge is that the practitioners were not aware of the assumptions when they made them, which is mainly due to their unfamiliarity of the AA concept. Two respondents stated that “Making assumptions implicitly without being aware of them” is a reason for not identifying AA, and four respondents considered it as the same impediment for not recording AA. We argue that the characteristics of practitioners have a significant impact on the identification and recording of AA. For example, if an architect does not care about AA or is even unfamiliar with this concept, s/he may not spend time and effort on identifying and recording AA in projects. To address this problem, it is viable to educate practitioners by e.g., tutorials to understand the AA concept, its importance, and become aware of the approaches, tools, and guidelines for AA identification and recording, as well as their costs and benefits.

- (4) Others (mentioned as either challenges or impediments)

The lack of support from specific stakeholders (such as customers and team leaders) in a project was regarded as an important challenge and obstacle in identifying and recording AA. This is related to the finding that the AA concept is not commonly understood in practice as discussed in [Section 5.1](#). For example, a project manager would not support the architect in her/his team to identify and record AA in a project, and regard it as waste of time, if the manager has no idea of what AA is and the importance of AA. To address this challenge, it is necessary and beneficial to present the cost-benefit analysis of AA identification and recording to all stakeholders (e.g., project manager) before the development.

The lack of support from AA experts is another important challenge. It would be difficult for the development team to identify and record AA, if no one is familiar with the AA concept and its management. To address this challenge, the education of AA mentioned in point (3) of this section, and frequent communication between involved stakeholders and AA experts (e.g., senior architects) are encouraged.

Additionally, we argue that the evaluation of the costs and benefits of identifying and recording AA should be another important impediment: practitioners are rightfully reluctant to perform activities that do not have a clear and favorable cost-benefit ratio. To address this impediment, we need to conduct more empirical studies (e.g., surveys and case studies) and provide evidential data for the cost-benefit analysis of AA identification and recording in development.

5.3. Approaches and tools

Our results show that there is a gap between how AA is regarded in industrial software development and how it is supported. On the one hand, most respondents considered that AA are important in both software architecting and the software development lifecycle (see [Section 4.3](#)); on the other hand, there is a lack of approaches and tools to identify and record AA (see [Sections 4.4 and 4.5](#)). This gap can be the motivation for researchers and practitioners to investigate, develop, and practice more suitable and dedicated approaches and tools for AA identification and recording.

24.5% of the respondents (12 out of 49) identified AA based on their personal experience and knowledge (in [Table 7](#)). Like in most software engineering activities, prior experience and knowledge can be a significant factor in dealing with AA ([de Graaf et al., 2014](#)). Future approaches for managing AA should take this factor

into account in order to support stakeholders with a varying level of experience and knowledge.

The three tools that the respondents reported for AA identification and the six tools for AA recording are general tools in software development, and by no means dedicated for the purpose of identifying or recording AA. It is important to understand the needs of practitioners when identifying and recording AA in various development context, and provide dedicated tools to support these two AA activities. As mentioned in Section 5.2, “Time constraint of projects” was identified as both challenges and impediments in AA identification and recording. We can then conclude that the tools for AA identification and recording should be lightweight, well-integrated with existing tools, and provide just-in-time support for the AA activities without the necessity of a priori effort, in order to alleviate the time pressure in development.

Identifying and recording AA can be either performed independently in software development or integrated with the management of other artifacts (e.g., architectural design decisions) depending on the understanding of AA. Further research on AA identification and recording approaches and tools with which AA identification and recording can provide more benefits than costs would be beneficial. For example, a project with limited resources may afford to spend only little effort on AA identification but still expect some benefit, thus requiring a lightweight approach.

5.4. Impact of the characteristics of respondents

In order to understand the impact of the characteristics of respondents to the survey results, we classified the respondents into two groups according to their main tasks in development (DQ3) and their experience in software architecting (DQ5).

(1) Main tasks in development

The respondents who chose the tasks “Architecture design” or “Detailed design” as their main tasks in development (see Fig. 2), have more frequently used the AA term (SQ2, 15.0% vs. 6.3%), and identified (SQ8, 50.0% vs. 28.1%) and recorded (SQ12, 21.3% vs. 15.6%) AA than the respondents who chose other tasks (e.g., testing). This comparison indicates that the architects and designers are more likely to be familiar with the concept of AA and use it in their everyday practice.

(2) Experience in software architecting

The respondents who have at least 3 years of experience in software architecting are more familiar with the AA term (SQ1, 39.0% vs. 26.4%), more frequently used the AA term (SQ2, 16.9% vs. 7.5%), and identified (SQ8, 59.3% vs. 26.4%) and recorded (SQ12, 23.7% vs. 15.1%) AA than the respondents with less than 3 years of experience in software architecting. This comparison implies that experience in software architecting is an important factor that impacts understanding, identifying, and recording AA.

5.5. Attitude and location of target population

More than 900 visitors entered the questionnaire system, and 213 of them (about 24%) completed the survey until the end. We do not know at which specific question visitors stopped and dropped out, because the questionnaire system that hosts this survey does not support this function. It is not a surprising result that less than one-quarter visitors finished the questionnaire; in a survey for a similar topic (architecture design rationale), a similar response rate (about 23%) was observed (Tang et al., 2006). Most visitors we invited came from the three technical websites presented

in Section 3.2.2. These visitors have the title of “architect”, wrote SA blogs, or just show an interest in SA, but we have no comprehensive information about their expertise or their role(s) in their companies. Because of this limited personal information, it is possible that we sent the invitation to people without much interest in the topic of AA. We list other four possible reasons for the low response rate: (1) the same person may enter the questionnaire system several times. For example, the authors had to enter the system to check the questionnaire; (2) the visitor had little knowledge of software architecture or design (e.g., had no idea of the AA concept), and the questionnaire was difficult for the visitor to understand; (3) the visitor lost patience when answering certain questions; (4) 10–15 min for finishing the questionnaire was still too long for the visitor.

As mentioned in Section 4.1, 101 (47.4%) invalid responses were excluded based on the selection criteria. We list two possible reasons for this high percentage: (1) the respondent had no interest in this topic (AA) or lost patience when answering certain questions, and answered the questions carelessly (e.g., answer like “aaa” or “...”); (2) the respondent had little knowledge of software architecture or design (e.g., had no idea of the AA concept), and they just finished the questionnaire by clicking next.

Beijing (24 out of 112, 21.4%), Shenzhen (21 out of 112, 18.8%), Shanghai (11 out of 112, 9.8%), and Guangzhou (9 out of 112, 8.0%) are the four leading cities in the survey (in Table 3). This is reasonable as these four cities are the most developed cities of China (Shen and Yang, 2014). Note that no respondents come from Hong Kong, Macao, and Taiwan. The main reasons are that: (1) the three websites used for inviting potential participants are not the major media for the practitioners in these three regions; (2) the social networks that we used mainly target users from mainland China; (3) the International Architect Summit in Beijing primarily attracts participants from mainland China.

6. Threats to validity

The threats to the validity of the survey results are discussed in this section according to the guidelines by Wohlin et al., (2012). Internal validity is not discussed as this survey did not deal with causality between variables and the survey results.

Construct validity in a survey focuses on whether the survey constructs are defined and interpreted correctly (Ding et al., 2014).

AA is the core construct of the survey. According to the results of this survey, there is no common understanding among practitioners about the concept of AA, only 33.0% (37 out of 112) respondents were familiar with the AA term, and only 12.5% (14 out of 112) respondents used the term in their projects, which may lead to a threat of AA misinterpretation. However, we can still conclude that most of the respondents had a fair understanding of the AA concept based on the points below:

- (1) AA term is not the same as the AA concept. It is possible that a respondent is not familiar with the AA term and never used it, but has a fair understanding of the AA concept. Evidence can be found in the answers of SQ4 and SQ5: most of the respondents came up with meaningful AA examples (73.2%, 82 out of 112, respondents gave at least one meaningful example of AA), which is an evidence to support that most of the respondents understood the AA concept.
- (2) We conducted three interviews in the pilot study and tried to find a term of AA in Chinese commonly used by practitioners. The interviewees told us that they could understand the AA term we proposed, and they believed that the AA concept does exist in software development and architecting. But all the three interviewees (Chinese) could not provide a better term than the one we proposed.

- (3) We provided an example of assumption from everyday life (*when reserving a wedding party, you assumed that 80% of the total 100 people who were invited would attend the party, and based on this assumption, you decided to book 7 tables and 1 backup table in the restaurant, see Appendix*) at the welcome page of the survey. By providing this non-technical example of an assumption, we intended to help the respondents to better understand the AA term with their architecting experience (most of the respondents, 97.3%, 109 out of 112, have experience in software architecting); furthermore a non-technical example of an assumption would not bias the respondents from providing their own understanding of AA.

It is possible that some respondents might have used AA in their projects, but answered “No” to the specific questions (e.g., SQ4 and SQ5) simply because they never heard the AA term we proposed. This may have resulted in missing some valid answers from those respondents.

Another threat concerns whether questions can be clearly understood. To mitigate this threat, the survey protocol was reviewed by all the authors and two colleagues, and we made a pilot study with three interviews to examine their understanding of the survey questions.

We also considered that the questions may not be truthfully answered by the respondents (van Heesch and Avgeriou, 2011). To minimize this threat, we limited participation to a strictly voluntary and anonymous basis. However, this threat cannot be fully eliminated (van Heesch and Avgeriou, 2011).

Finally, since all the responses were written in Chinese, the translation of the raw data from Chinese to English may have resulted in lost or corrupted information. To minimize this threat, two authors who are native Chinese speakers translated the raw data together, and discussed the translation results with the third author to make sure the translation was as precise as possible.

External validity refers to the degree to which the findings from this survey can be generalized (Ding et al., 2014). There is always a risk in surveys that the sample is biased (not being representative of the target population) when using a Non-Probabilistic Sampling method. To mitigate this threat, besides asking the contacts of our social networks, three popular technical websites and one international meeting of software architecture (International Architect Summit) were used to ensure that we attracted participants from diverse backgrounds and organizations. Thus we can assert that the results are representative for Chinese practitioners with profiles similar to those described in Section 3.2.2. It is possible that the results are generalizable to other countries too, but this can only be confirmed by replicating this study.

Reliability focuses on whether the survey would yield the same results if other researchers replicated it (Ding et al., 2014). To mitigate this threat, two authors analyzed the data independently, and discussed the analysis results with the third author in case of conflicts until an agreement was reached. Furthermore, reliability can be affected due to a different understanding by different researchers of the inclusion and exclusion criteria for selecting responses (see Section 3.2.3). To minimize this potential bias of researchers, the inclusion and exclusion criteria were designed, discussed, and verified in a pilot study (seven respondents with three interviews). Finally, different understandings by different researchers of the answers to the survey questions may lead to a threat of misinterpretation of these answers. To mitigate this threat, the authors discussed the survey data and reached a consensus on the understanding of the answers to the survey questions.

7. Conclusions and future work

We gained insight on the topic of AA, regarding how practitioners understand AA and its importance, and whether and how they identify and record AA in software development. The survey is interesting for practitioners who are interested in identifying and recording AA in their projects, as well as for researchers who are interested in exploring the understanding of AA and providing support for identifying and recording AA in practice. We analyzed 112 valid responses and the main findings are as follows:

- (1) AA are important in both software architecting and the software development lifecycle, and they are worth to be managed (e.g., identifying and recording) in projects. However, practitioners understand AA in different ways, and use different terms to represent the AA concept.
- (2) The concept of AA is regarded in five different ways that can be classified in corresponding categories (i.e., “Requirement”, “Architecture”, “Constraint”, “Risk”, and “Project context”).
- (3) Only a few respondents identified and recorded AA in their projects, and very few approaches and tools were used for identifying and recording AA.
- (4) The results of this survey show that the challenges of identifying and recording AA and the reasons for not identifying and recording AA are largely overlapping. The most significant challenges (reasons) are the lack of dedicated approaches and tools for AA identification and recording.
- (5) There is a discrepancy in how AA are treated in software development. On the one hand, most respondents considered that AA are important in both software architecting and the software development lifecycle; on the other hand, there is a lack of approaches and tools used in practice to identify and record AA.

To address the challenges of identifying and recording AA as well as the reasons for not identifying and recording AA, we proposed four future directions:

- (1) To develop an AA documentation framework for identifying and recording AA.
- (2) To develop a systematic process for AA management (including identifying and recording AA) embedded in the development process (e.g., Iterative development), with the support of specific approaches and dedicated tools, especially lightweight ones.
- (3) To develop guidelines for understanding the AA concept and managing AA, e.g. how to introduce AA management tailored to individual projects, which AA need to be managed, who should be involved in AA management and when.
- (4) To identify the factors that may impact AA management. For example, the characteristics (e.g., knowledge, experience, and preference) of practitioners are potential factors that impact AA management.

Acknowledgment

This work is partially sponsored by the NSFC under Grant nos. 61170025 and 61472286 and the Ubbo Emmius scholarship program by the University of Groningen.

Appendix

Tables 12–14

Table 12

The welcome page of the survey.

This survey aims to investigate the concept of Architectural Assumption in industrial practice, in order to align research efforts towards supporting practitioners to better manage architectural assumptions.

An example of assumption from everyday life: *when reserving a wedding party, you assumed that 80% of the total 100 people who were invited would attend the party, and based on this assumption, you decided to book 7 tables and 1 spare table in the restaurant.*

All the personal information collected in this survey will be kept confidential. Furthermore, the report derived from this survey will be available for all participants that wish to receive it.

Answering the survey may take about 15–20 min. Thanks a lot in advance for your support!

Table 13

Questions on demographics of the survey.

ID	Questions on demographics	Type of answers
DQ1	Which country are you working in?	Free text
DQ2	What is your educational background (highest degree obtained)?	BSc / MSc / PhD / Others
DQ3	(Multiple Choice) What are your main tasks in your company?	Project management / Requirements elicitation & analysis / Architecture design / Detailed design / Coding / Testing / Others
DQ4	What is your experience (in years) working in IT industry?	Free text (Integer ≥ 0)
DQ5	What is your experience (in years) in software architecting?	Free text (Integer ≥ 0)
DQ6	Have you ever received any professional training (i.e. excluding higher education) related to software architecture or software design?	Yes / No
DQ7	What is the size of your company (number of employees)?	<10 / 10–50 / 50–250 / > 250
DQ8	(Multiple choice) What is the domain of your company?	IT services / Embedded system / E-commerce / Financial / Healthcare / Telecommunication / Retail / Insurance / Other domains
DQ9	What development methods were (commonly) used in your projects?	Free text

Table 14

Specific questions of the survey.

ID	Specific questions	Type of answers
SQ1	Are you familiar with the term Architectural Assumption?	Yes / No
SQ2	Have you ever used this term in your work?	Yes / No
SQ3	How did you use this term in your work?	Free text
SQ4	(Optional Question) Can you provide your definition of architectural assumption (brief description in 1–3 sentences)?	Free text
SQ5	Can you provide an example of architectural assumption (or examples if you think there are more than one types) according to your understanding?	Free text
SQ6	How would you grade the importance of architectural assumptions in software architecting?	Unimportant/Of little importance/Moderately important/Important/Very important/No idea
SQ7	How would you grade the importance of architectural assumptions in the software development lifecycle (i.e. from requirements analysis to software maintenance)?	Unimportant/Of little importance/Moderately important/Important/Very important/No idea
SQ8	Have you ever identified architectural assumptions (e.g., recognized existing architectural assumptions or came up with new architectural assumptions) in your projects (e.g., in architecture design)?	Yes/No
SQ9	How did you identify architectural assumptions in your projects? If an approach and/or a supporting tool were used, can you briefly describe them?	Free text
SQ10	(Multiple Choice) During the identification of architectural assumptions, what kind of challenges did you face (if any)?	Lack of management support/Lack of approaches/Lack of tools/Lack of time/Lack of experts of architectural assumptions/Lack of guidance/Making assumptions implicitly without being aware of them/Other reasons

(continued on next page)

Table 14 (continued)

ID	Specific questions	Type of answers
SQ11	(Multiple Choice) What are the major reasons that hindered you from identifying architectural assumptions in your projects?	No time/No benefit/Costs outweigh benefits/Lack of approaches/Lack of tools/Other reasons
SQ12	Have you ever recorded architectural assumptions (e.g., explicitly described architectural assumptions in a document or a wiki) in your projects?	Yes/No
SQ13	How did you record architectural assumptions in your projects? If an approach and/or a supporting tool were used, can you briefly describe them?	Free text
SQ14	(Multiple Choice) During the recording of architectural assumptions, what kind of challenges did you face (if any)?	Lack of management support/Lack of approaches/Lack of tools/Lack of time/Lack of experts of architectural assumptions/Lack of guidance/Making assumptions implicitly without being aware of them/Other reasons
SQ15	(Multiple Choice) What are the major reasons that hindered you from recording architectural assumptions in your projects?	No time/No benefit/Costs outweigh benefits/Lack of approaches/Lack of tools/Other reasons

References

- Adolph, S., Hall, W., Kruchten, P., 2011. Using grounded theory to study the experience of software development. *Empir. Softw. Eng.* 16 (4), 487–513.
- Corbin, J., Strauss, A., 2014. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, 4th Edition Sage Publications.
- Ding, W., Liang, P., Tang, A., van Vliet, H., Shahin, M., 2014. 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.
- Del Greco, L., Walop, W., 1987. Questionnaire development: 1. Formulation. *Can. Med. Assoc. J.* 136 (6), 583–585.
- de Graaf, K.A., Liang, P., Tang, A., van Vliet, H., 2014. The impact of prior knowledge on searching in software documentation. In: *Proceedings of the 6th ACM Symposium on Document Engineering (DocEng)*. Fort Collins, Colorado, USA, pp. 189–198.
- Easterbrook, S., Singer, J., Storey, M.A., Damiani, D., 2008. *Selecting empirical methods for software engineering research. Guide to Advanced Empirical Software Engineering*. Springer, London, pp. 285–311.
- Fink, A., 1995. *The Survey Handbook*. Sage Publications, Thousand Oaks.
- Garlan, D., Allen, R., Ockerbloom, J., 2009. Architectural mismatch: Why reuse is still so hard. *IEEE Softw.* 26 (4), 66–69.
- Haley, C.B., Laney, R.C., Moffett, J.D., Nuseibeh, B., 2006. Using trust assumptions with security requirements. *Requir. Eng.* 11 (2), 138–151.
- ISO/IEC/IEEE Std 42010-2011, *Systems and software engineering – Architecture description*, 2011.
- Kitchenham, B.A., Pfleeger, S.L., 2008. Personal Opinion Surveys, in *Guide to Advanced Empirical Software Engineering*. Springer, pp. 63–92.
- Larman, C., 2004. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional.
- Larman, C., Basili, V.R., 2003. Iterative and incremental development: A brief history. *IEEE Comput.* 36 (6), 47–56.
- 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.
- Lethbridge, T.C., 2005. Studying software engineers: data collection techniques for software field studies. *Empir. Softw. Eng.* 10 (1), 311–341.
- Lewis, G.A., Mahatham, T., Wragge, L., 2004. Assumptions management in software development. CMU/SEI-2004-TN-021 Technical Report.
- Lehman, M.M., Ramil, J.F., 2001. Rules and tools for software evolution planning and management. *Ann. Softw. Eng.* 11 (1), 15–44.
- 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.
- Shen, J., Yang, X., 2014. Analyzing urban competitiveness changes in major Chinese cities 1995–2008. *Appl. Sp. Anal. Policy* 7 (4), 361–379.
- Tang, A., Babar, M.A., Gorton, I., Han, J., 2006. A survey of architecture design rationale. *J. Syst. Softw.* 79 (12), 1792–1804.
- Tang, A., Jin, Y., Han, J., 2007. A rationale-based architecture model for design traceability and reasoning. *J. Syst. Softw.* 80 (6), 918–934.
- 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.
- van Heesch, U., Avgeriou, P., 2011. Mature architecting – a survey about the reasoning process of professional architects. In: *Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. Boulder, Colorado, USA, pp. 260–269.
- 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.
- Vanthienen, J., Wets, G., 1994. From decision tables to expert system shells. *Data & Knowl. Eng.* 13 (3), 265–282.
- Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. *Experimentation in Software Engineering*. Springer.
- 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.
- 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 Ph.D. student, he had worked as a senior software engineer and architect in industry for four years.

Peng Liang is a 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 Youth Associate Editor of *Frontiers of Computer Science*, Springer. He has published more than 70 articles in peer-reviewed international journals, conference and workshop proceedings, and books.

Paris Avgeriou is a professor of software engineering in the Department of Mathematics and Computing Science, University of Groningen, the Netherlands where he has led the software engineering research group since 2006. He has co-organized several international conferences and workshops and sits on the editorial board of *Springer TPLP*. He has edited special issues in *IEEE Software*, *Elsevier JSS* and *Springer TPLP*. He has published more than 120 peer-reviewed articles in international journals, conference proceedings and books. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution and patterns.