Identifying and Recording Software Architectural Assumptions in Agile Development

Chen Yang  
State Key Lab of Software Engineering  
School of Computer, Wuhan University  
Wuhan, China  
cyang@whu.edu.cn

Peng Liang*  
State Key Lab of Software Engineering  
School of Computer, Wuhan University  
Wuhan, China  
liangp@whu.edu.cn

Abstract—Architects and involved stakeholders constantly make Architectural Assumptions (AAs) in architecture design. These assumptions, as an important part of architectural knowledge, need to be well managed in the whole architecting lifecycle. However, they are always retained in the heads of various stakeholders and left undocumented, which results in architectural knowledge vaporization, especially in agile development when documentation is not a must and priority. In this paper, we propose a novel method that is composed of two parts – Architectural Assumption Library (AAL) to help architects identify AAs and Architectural Assumption Card (AAC) to record them based on a simplified conceptual model of AA. An architect from an industry project of Embedded Systems that using agile development was invited to employ this method in the project to identify and record AAs. It shows that the proposed method can help architects to easily identify and record AAs, which is promising to facilitate architecture design, maintenance, and evolution in agile development.

Keywords—software architecture; architectural assumption; agile development

I. INTRODUCTION

Software architecture (SA) presents a high-level design of a system, which has an important role to manage complicated interactions between stakeholders of software-intensive systems in order to balance all kinds of constraints [1]. There are many definitions of SA. Bass et al. define SA as the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both [1]. In a design decision perspective, a software architecture is composed of a set of architectural design decisions (ADDs), which lead to the resulting design artifacts [2]. Traditional architecting process tends to employ big design up-front (BDUF) leading to massive documentation and implementation of YAGNI (you ain’t gonna need it) features, which introduces redundancy in architecting as well as development effort [3].

Agility means to strip away as much of the effort-intensive activities in software development, commonly associated with the traditional software development methodologies, as possible to promote quick response to changing environments, changes in user requirements, accelerated project deadlines and the like [4]. Augustine notes that the common characteristics of agile development methods are iterative and incremental lifecycle, which focus on small releases, collocated teams, and a planning strategy based on two levels: release plan driven by a feature or product backlog, and an iteration plan handling a task backlog [5]. These common characteristics adhere to the values of the agile manifesto [6].

It is challenging to apply SA in agile development. Proponents of agile approaches usually see little value for system’s customers in the up-front design and evaluation of architecture [3]. Agile development on the contrary focuses on enough design till the last responsible moment which can lead to simplicity in development and a rapid and iterative process of producing artifacts [8]. To combine the benefits from both approaches (agility and architecture), a transformation from traditional SA to agile SA is emerging [8]. In an architecting process, numerous implicit and undocumented ADDs are more or less arbitrarily made on the fly based on personal experience, domain knowledge, budget constraints, available expertise, and the like [9]. Some researchers called these decisions as architectural assumptions (AAs) [9][10], which are an important part of architectural knowledge to be managed during the entire software lifecycle [11]. Therefore, how to identify and record AAs in agile development with a lightweight approach is a challenging issue to align architecture and agile development [8].

It worth noting that ISO 42010 standard [12] provides a conceptual model for architecture description, but this standard is too heavyweight to document SA in an agile environment and considers AA as an implicit component of certain standard elements, such as Concern, rather than an explicit element that should be documented. We tried to address this limitation by proposing a simplified model for recording AAs in this work.

In agile development, AAs are usually made implicitly. For example, a system needs high security, and we asked the architect of the system why not consider the external security (such as broken access control, cross-site scripting, etc.) of the system (we didn’t find any information about the external security of the system in the SA documents) when he designed the architecture. He explained that: “the system will be deployed in an internal environment which is secure enough, so we do not need any other mechanisms to achieve security.
This decision can avoid some redundant development work.”

This is a typical example of AA in software development without any documentation but stored in stakeholders’ (e.g., architect) mind. Does this AA fit the requirement of customers? What are the pros and cons of making this AA? It is difficult to answer such questions, since the AA was implicit and only a few stakeholders knew it, especially when the knowledge about the AA was vaporized gradually.

Implicit and undocumented AAs can lead to some problems such as misunderstanding of architecture, design violation, etc. It may be difficult to decide whether an assumption is an appropriate one for the current development context, but it is beneficial to make AAs explicit. Therefore, we need to manage AAs in an explicit and lightweight way, which can make AAs play an active and important role in agile development lifecycle.

The rest of this paper is organized as follows. Section II summarizes related work. In Section III, we introduce AA and propose a simplified conceptual model of AA. In Section IV, we propose a novel method – Architectural Assumption Library and Architectural Assumption Card which is based on the simplified conceptual model in Section III – to identify and record AAs. Section V presents a case study of using the proposed method in an industry agile project. Section VI concludes this work with future work directions.

II. RELATED WORK

A. Assumption in Software Development

The concept of assumption is not new. There are many type of assumptions in software development, such as requirement assumption, architectural assumption, etc., which focus on different aspects of software development [10]. Related work on assumptions in software development covers the areas of their classification, impact, and the methods of processing them (such as recovering, identifying, and documenting assumptions).

In [10], a prototype Assumptions Management System in software development was developed by Lewis et al., which can extract assumptions from Java source code and record them into a repository for management. They provided a taxonomy of general assumptions in software development, including control assumptions (expected control flow), environment assumptions (expected environmental factors), data assumptions (expected input or output data), usage assumptions (expected use of applications), convention assumptions (followed standards or conventions in development).

B. Architectural Assumption

In [13], Garlan et al. identified four general categories for architectural assumptions that are implicit and undocumented and consequently lead to architectural mismatch: nature of components, nature of connectors, global architectural structure, and software construction process. This categorization of architectural assumptions is based on a structural view of architecture, which considers SA as a set of structures, including components and connectors.

In [11], Ireo and Michel proposed a lightweight method to manage architectural assumptions in agile development. They summarized four main tasks of architectural assumptions management from existing literature: recording new assumptions, monitoring assumptions on a regular basis, searching for assumptions, and recovering past assumptions. They use the taxonomy of assumptions proposed in [14]. Our method focuses on identifying and recording AAs based on a simplified conceptual model in agile development, which covers the task “recording new assumptions” in [11].

In [9], Roeller et al. classified architectural assumptions into four groups as: implicit and undocumented (the architect is unaware of the assumption, or it concerns of course knowledge), explicit but undocumented (the architect takes a decision for a very specific reason), explicit and explicitly undocumented (the reasoning is hidden), explicit and documented (this is the preferred, but often exceptional, situation). They also developed a method (Recovering Architectural Assumptions Method, RAAM) to recover assumptions in architecture design.

In [14], Patricia and Hans distinguish architectural assumptions from requirements and constraints. An assumption meta-model was provided to document these assumptions in an explicit way. They classified architectural assumptions into three groups: managerial, organizational, and technical assumptions. However, it is complicated and may not be appropriate to use this meta-model and classification of xxx assumptions in agile development. Our method of identifying and recording assumptions is lightweight to accommodate agile development and focuses on architectural assumptions.

C. Knowledge Management in Software Development

In [15], Rus and Lindvall described the knowledge evolution cycle in software development, which is composed of five phases: (1) Originate/create knowledge (members of an organization develop knowledge through learning, problem solving, innovation, creativity, and importation from outside sources); (2) Capture/acquire knowledge (members acquire and capture information about knowledge in explicit forms); (3) Transform/organize knowledge (organizations organize, transform, or include knowledge in written material and knowledge bases); (4) Deploy/access knowledge (organizations distribute knowledge through education, training programs, automated knowledge-based systems, or expert networks); (5) Apply knowledge (organizations apply the knowledge - which is the ultimate goal of knowledge management and aims to make knowledge available whenever it is needed). Our proposed method covers all these five phases of architectural knowledge management through identifying and recording architectural assumptions (a type of architectural knowledge) [7].

III. ARCHITECTURAL ASSUMPTION

We follow the definition of architectural assumption by Roeller et al. which regards AA as a kind of architectural design decision [9]. These decisions, as well as the reasons for making the decisions, are often not explicit and usually remain undocumented.
AAs are a kind of assumptions which focus on architecture design and they are an important type of architectural knowledge to be managed during the entire software development lifecycle [7]. AAs are always ignored and violated if they are not appropriately identified and recorded since these assumptions can fail at any time with serious consequences (e.g., misunderstanding of architecture, design violation, project delay, and low quality) if not being managed (identified and explicitly recorded) appropriately.

According to the four groups of AAs identified in [9]: implicit and undocumented, explicit but undocumented, explicit and explicitly undocumented, explicit and documented. We argue that most AAs in agile development are implicit and undocumented since documentation is not a must and priority [6]. Managing all the AAs in software lifecycle would cost a huge amount of resources which is not advocated and suitable in agile development. Therefore, AAs need to be managed in a lightweight manner. In this paper, we propose to identify and record AAs in agile development with a simplified conceptual model of AA as shown in Fig. 1.

This conceptual model is originated from the template for recording ADDs in [16]. The template has 14 elements: Issue, Decision, Status, Group, Assumptions, Constraints, Positions, Argument, Implications, Related decisions, Related Requirements, Related artifacts, Related principles, and Notes. Compared to the ADD template, our simplified AA conceptual model only contains 5 elements: “Stakeholder” element can improve the traceability of AAs to related stakeholders, who are the source of this AA. “Description” element provides basic information about what the AA is. “Pros” and “Cons” support stakeholders to understand the positive and negative impact when considering the AA. “Relationship” between AAs provides traceability information between AAs which is needed in architecture impact analysis [17]. We argue that these five AA elements are the core and fundamental elements for identifying and recording AAs in agile development, which is also largely identified as the major elements (assumptions, arguments) in ADD conceptual models [18].

**Fig. 1. A simplified conceptual model of Architectural Assumption**

### IV. Method

We propose a novel method, which is composed of two parts – Architectural Assumption Library (AAL) and Architectural Assumption Card (AAC) – for identifying and recording AAs in agile development. These two parts are both lightweight and based on the simplified conceptual model of AA proposed in Section III. A process of identifying and recording AAs is specified to connect these two parts, which is described in detail in this section.

#### A. Architectural Assumption Library

Identifying AAs usually cause extra effort in development. Roeller et al. proposed to use interviews to identify AAs [9]. However, it is overcomplicated for agile development. For example, architects and developers may spend several days to discuss with stakeholders and identify AAs in several iterations in one release [9].

An Architectural Assumption Library acts as a repository of AAs, and helps architects to reduce the effort of identifying AAs in agile development by providing existing and reusable AAs as a reference and starting point, which can also reduce the time and identification effort on the discussion of AAs among stakeholders. There are two activities in the AAL management: creation and maintenance of AAL. The creation activity includes initializing an AAL and adding AAs into the AAL, for example, the new AAs can be identified, recorded, and added during project development. The maintenance activity of AAL includes checking and removing duplicated AAs in an AAL, clarifying AAs description and their relationships, etc. In an agile development, time and human resources are precious and it may not be appropriate to maintain the AAL during the development, because for example, checking and removing duplicated AAs should be performed by architects, which would cost additional effort but has little value to the current development. We propose that architects can work with stakeholders to maintain AAL at the end of a project which is time wise and when they still have a clear memory about the AAs of the project.

We use the simplified conceptual model of AA in Fig. 1 as a basic template to record and add AAs into AAL. The core items of the template are described in TABLE I.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumption Name</td>
<td>This element describes the name of the AA. It should be meaningful to stakeholders, which means that it should not be named as “1”, “A”, etc. A meaningful name provides hint for users to quickly identify the AA.</td>
</tr>
<tr>
<td>Stakeholders</td>
<td>This element describes the stakeholders who are concerned about the AA. It provides the possibility to discuss the AA with the involved stakeholders to get a better understanding of the AA or modify this AA.</td>
</tr>
<tr>
<td>Description</td>
<td>This element provides more detailed information about the AA which helps stakeholders to understand the assumption.</td>
</tr>
<tr>
<td>Pros</td>
<td>This element helps stakeholders to get a clear understanding of the positive impact when considering the AA.</td>
</tr>
<tr>
<td>Cons</td>
<td>This element helps stakeholders to get a clear understanding of the negative impact when considering the AA.</td>
</tr>
<tr>
<td>Relationship(s)</td>
<td>This element specifies the relationship between AAs. Kruchten provided a taxonomy for ADD in [19]. We adapt this taxonomy for AA with</td>
</tr>
</tbody>
</table>
minor change. TABLE II presents the five relationships between AAs.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Constrains B</td>
<td>Architectural assumption B is tied to assumption A. If A is removed, then B should be removed as well.</td>
</tr>
<tr>
<td>A Conflicts with B</td>
<td>A symmetrical relationship indicating that the two architectural assumptions A and B are mutually exclusive.</td>
</tr>
<tr>
<td>A Comprises B₁, B₂, …, Bₙ</td>
<td>A high level and somewhat complicated, wide-ranging architectural assumption A is made of or decomposed into a series of narrower, more specific architectural assumptions B₁, B₂, …, Bₙ.</td>
</tr>
<tr>
<td>A Is an alternative to B</td>
<td>A and B are similar architectural assumptions, and they can replace each other.</td>
</tr>
<tr>
<td>A Is related to B</td>
<td>There is a relation of some sort between the two architectural assumptions, but it is not of any kind listed above.</td>
</tr>
</tbody>
</table>

### B. Architectural Assumption Card

We provide Architectural Assumption Card based on the simplified conceptual model of AA as well (as shown in Fig. 1), to record AAs in agile development. These recorded AAs will be shared among all stakeholders of a project using blackboard or collaborative tools. An example AA recorded using AAC is provided in TABLE III. By using AAC, AAs can be recorded in an explicit and lightweight manner, which helps involved stakeholders to communicate the architecture design related to certain AAs.

#### TABLE III. An Example of AA Recorded Using Architectural Assumption Card

<table>
<thead>
<tr>
<th>AAC Element</th>
<th>Content of Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumption Name</td>
<td>Response time of the system</td>
</tr>
<tr>
<td>Stakeholders</td>
<td>Jun Lan Yang (architect)</td>
</tr>
<tr>
<td>Assumption Description</td>
<td>The response time of the system should be within 2 seconds</td>
</tr>
<tr>
<td>Pros</td>
<td>High performance and usability</td>
</tr>
<tr>
<td>Cons</td>
<td>Extra effort (such as testing, hardware, etc.)</td>
</tr>
<tr>
<td>Relationship(s)</td>
<td>Is constrained by Response strategy</td>
</tr>
</tbody>
</table>

### C. Process of Identifying and Recording Architectural Assumptions

This process comprises three parts – using AAL to identify AAs, using AAC to record AAs, and adding new AAs into AAL (which is part of the creation activity in AAL management as discussed in Section IV.A). Fig. 2 shows the process of identifying and recording AAs using AAL and AAC, which is composed of five steps:

1. Use search strategies, such as keywords, to search related AAs in AAL;
2. If result AAs are retrieved, then identify the AA which is usable for the current design context. Otherwise, go to Step (4);
3. If the identified AA does not directly suit the current design context, then this AA should be adapted. Otherwise, go to Step (4);
4. Record the AA using AAC. The AA recorded can be both an AA identified and adapted in AAL or a new AA from the current project;
5. If the AA is new, then add it into AAL.

![Fig. 2. Process of using AAL and AAC to identify and record AAs](image)

### V. Case Study

To evaluate our proposed method in an industrial context, we invited an architect from an industry project on Embedded Systems that employing agile development to use the simplified AA conceptual model and the process in the project to identify and record AAs. The collected AAs are limited because there is only one project involved, and we plan to further evaluate this method empirically with e.g., controlled experiments in the next step.

This project is an embedded system based on the Internet of Things techniques for managing fixtures in a big clothing factory. One fixture can be possibly moved on demand to everywhere in the factory which is composed of several workshops, and factory administration needs to find out where it is and its trajectory when they need it. They used readers, locators, electronic tags, and computers to build the hardware environment. The operation principles of the embedded system are described below:

1. Every fixture has a tag, providing its basic information, including a unique identity ID.
(2) Every workshop has been divided into several areas and each of them has a locator. Position information of this area has been stored in locators, and they can activate tags through sending data with 125KHz (the frequency used in Radio Frequency Identification) in a fixed period.

(3) Workshops have been divided and each of them has a reader. The reader can detect the tag and receive the data with position information from the tag.

(4) Through the wireless network, the readers can send the data received from the tags to the computer.

(5) After processing the data in the computer, users can retrieve the position information of the fixture.

We gave a short tutorial of AA to the architect of this embedded system, and asked him to construct an initial AAL using the template (as shown in TABLE I) based on his own experience from previous industry projects. More than 20 AAs were added in this initial AAL. During this project, the architect used the AAL to identify AAs and then recorded AAs using AAC. For example, the architect identified an architectural assumption “using 433MHz model (the communication frequency between readers and tags) with 40 meters of the communication distance” from the AAL, which is suitable for the current design context. Based on the actual environment (metal cover, signal interference, etc.) and his own experience, he adapted this AA into 433MHz model with 20 meters of the communication distance, and recorded this AA using AAC. The architect can use the AAs in AAL to reduce effort in architecture design. On the other hand, making these AAs explicit can help stakeholders to have a clear view in their perspectives: project managers and customers could know how many fixtures they need to cover the whole workshops; implementation consultant could figure out how to deploy the whole system; etc. Several examples of AA recorded by the architect using our method are listed in TABLE IV.

After the case study, we asked the architect to fill in a questionnaire to obtain a qualitative evaluation about our method. The questions and answers are shown in TABLE V.

The evaluation and feedback from the architect show that the process and model for identifying and recording AAs is easy to follow and takes acceptable effort to manage implicit AAs. The method has a positive impact on the development of project. He mentioned that some other activities related to AA are also important in agile development such as AA understanding, recovery, and evaluation. However, he suggested that we should consider the cost of introducing these AA activities into agile development and to provide appropriate tools to support the method. He also mentioned that whether this method can be suitable and readily used in an industrial setting depends on some factors, such as team organization, motivation of stakeholders, etc. Furthermore, he thought it was necessary to enact some shared strategies for AAs to help stakeholders easily apply the method in agile development.

According to his suggestions, we plan to classify AAs into some groups based on domain of projects or stakeholder role. This can be helpful to share AA information to the right stakeholders and reduce the redundancy of use (users may not want to read AAs they do not care).

VI. CONCLUSIONS AND FUTURE WORK

Architecture assumptions in agile development are normally retained in stakeholders’ mind which results in architectural knowledge vaporization [2]. It is important to manage them in an explicit way to avoid misunderstanding of architecture in architecting process. The major contributions of this work: (1) We propose that companies or organizations can build an Architectural Assumption Library to store their AAs which can be further used to identify AAs in agile development projects. (2) We propose to use Architectural Assumption Card to record AAs in agile development. Architectural assumptions can be explicitly written in a card or stored in collaborative tools to be better shared and communicated among stakeholders. (3) We propose a process of identifying and recording AAs using AAL and AAC with a simplified conceptual model of AA. (4) Finally, we report on an industrial case study, which was conducted to show the feasibility and value of our method in agile development.

We outline our future work in several points: (1) Empirically validate the proposed method in industrial agile projects using e.g., controlled experiments; (2) Provide supporting tools that help identify, record, and share AAs using the method; (3) Recover AAs and evaluate the quality of AAs which has a positive impact to the use of AAs in architecture design; (4) Classify AAs into different groups based on domain of projects or stakeholder role to easy their use in agile development.

REFERENCES


1. Do you clearly know what architectural assumptions are?

- Fully understand
- Somewhat understand
- Do not understand

2. Do you agree that architectural assumptions need to be managed in agile development?

- Yes
- No
- It depends

3. To what extent can you use our method to identify and record architectural assumptions in the agile project?

- Easy to use
- Need some effort but acceptable
- Difficult to use

4. Do you agree that the efficiency in agile development can be improved by identifying and recording architectural assumptions?

- Yes
- No
- I do not know

5. Do you agree that tools are necessary when identifying and recording architectural assumptions in agile development?

- Yes
- No
- I do not know

6. What other activities related to architectural assumptions do you think are important in agile development (such as architectural assumptions recovery, etc.)?

Answer: understanding, recovery, and evaluation of architectural assumptions

7. What are your comments and suggestions for improving the method based on your experience on agile development?

Answer:
(1) Whether this method can be suitable and readily used in industry is related to some other factors, such as team organization, motivation of stakeholders, etc.
(2) It is hard to define what types of architectural assumptions should be shared to what stakeholders, because we cannot and it is not necessary to share all the assumption information to everyone, but we only need to share the information to the specific stakeholders who need it. For example, it is necessary to share the system response time to the customers, but it may not be necessary to share the technical assumptions with them.