The Journal of Systems and Software 113 (2016) 362–380

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

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.

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 lifecycle. 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 (AA[1]), 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. 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.

http://dx.doi.org/10.1016/j.jss.2015.12.016
0164-1212/

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 recoding.
(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