Architectural Design Decisions in Open Software Development: A Transition to Software Ecosystems

Meiru Che, Dewayne E. Perry
Department of Electrical & Computer Engineering
The University of Texas at Austin
Austin, Texas, USA
meiruche@utexas.edu, perry@mail.utexas.edu

***Abstract*—Managing architectural design decisions (ADDs) in software development process is an essential task for architectural knowledge management. As software ecosystems become a new software development paradigm in software engineering processes, it is important and necessary to capture and represent ADDs in open software development, and to evolve architectural knowledge with minimum knowledge evaporation in the open ecosystem community. So far, little work has been done on managing architectural decisions in software ecosystems in current software architecture research and practice. In this research position paper, we discuss the typical characteristics of software ecosystems which may influence architecture decision-making processes in software development, and identify the essential aspects that should be considered for managing ADDs in the context of software ecosystem. In addition, we discuss major challenges of managing ADDs for software ecosystems, and we also propose possible directions in research to solve the problems.**

Keywords—architectural design decisions; open software development; software ecosystems; architectural knowledge;

I. Introduction

A recent strand of software architecture research is that where software architecture is considered as a set of architectural design decisions (ADDs) [1]. The specific focus on ADDs led to a broader focus on architectural knowledge [9]. Capturing and representing ADDs helps to organize architectural knowledge and reduce vaporization, and maintain consistency between requirements and the deployed system, therefore providing better control on many fundamental architectural drift and erosion problems in a software development life cycle [10]. An increasing number of models and tools are emerging to capture, manage, and share ADDs [12], [4], [7], [5].

A recent phenomenon in the evolution of software development strategies is that of encouraging external software developers to become involved in software development. These third parties make their contributions to software development and software organizations realize intrinsic benefits. This significant shift in the traditional software development process has resulted in a new software development paradigm called "software ecosystems". The adoption of the software ecosystem approaches establishes a new area in software engineering research and practice. Basically, in a software ecosystem, software organizations have broken their organization boundaries, and different parties collaborate under a common architecture and within a social networking context to achieve innovation. Therefore, the traditional *closed* software development has changed to *open* software development.

Current approaches to managing ADDs within a software organization for single product development may not be applicable for software ecosystems. The popularity of software ecosystems forces researchers and practitioners to reconsider how to manage architectural knowledge in open software development, since ADDs should be shared not only within the organization but also with external parties. Thus, a number of challenges of managing ADDs in a software ecosystem platform will arise, and it is important to develop approaches to managing architectural knowledge effectively in order to adapt the increasing openness and interoperability in the software community. So far, little work has been conducted in ADD management in software ecosystem research and practice.

In this research position paper, we firstly discuss the characteristics of software ecosystems, and then we briefly present our previous work on the approach to localized ADD documentation and evolution. Based on this, we discuss the essential aspects of managing ADDs in software ecosystems, and analyze and summarize the expected challenges. We also propose the possible directions in research to solve the problems at the end of the paper. To the best of our knowledge, our study is the first to provide challenges and directions on ADD management in software ecosystem contexts.

II. Software Ecosystems Characteristics

A software ecosystem is defined as a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them [8]. Compared with traditional software engineering, software ecosystems have the following characteristics:

A social community. In a software ecosystem, third parties are encouraged to contribute to an organization's product development, which establishes a social network that includes not only the team within the organization but also external developers, sharing technologies, skills, knowledge and even issues in the network. This further accelerates social interactions among the organization and the external parties, and forms a software social community.

Extensive business innovation. Innovations are always used to illustrate the capability of an organization to be creative in product development [3]. In a software ecosystem, both employees in the organization and the third parties have opportunities to provide innovative ideas to solve business

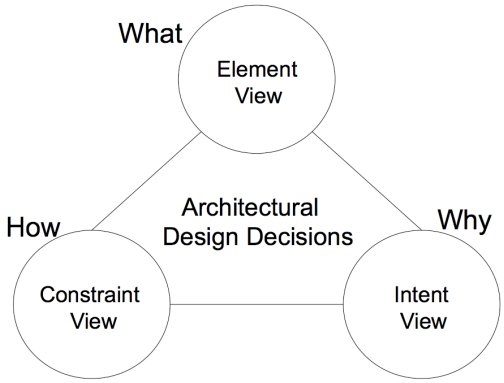

Figure 1. Triple View Model Framework

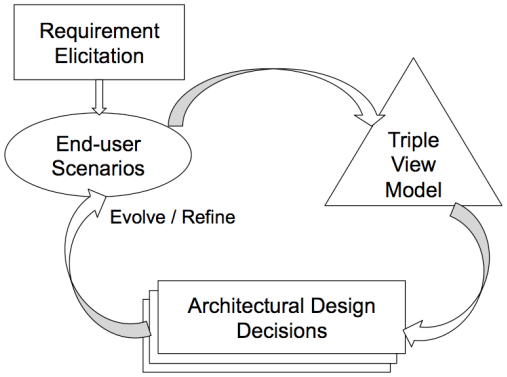

Figure 3. The Process of the Scenario-Based Method

problems, which extend the organization's innovative strategy and support both reactive and proactive business innovations [3].

Architecture platform commonality and variability. The concept of a software ecosystem focuses on multiple product development achieved by sharing a common architecture platform in open software development. However, software organizations also need to provide stable interfaces through the architecture platform to external developers, without disabling the operation of externally developed applications on top of the platform [2]. The concern of the architecture in a software ecosystem is to manage its commonality and variability to suit different business entities.

Diverse management. In a software ecosystem, both employees in an organization and external developers share community resources which include not only technical resources in the development but also tacit knowledge behind the thoughts of all parties. Thus, management is required to deal with the diverse resources distributed in multiple developments and multiple stakeholders, which influence the decision-making processes and the corresponding architectural knowledge.

III. Localized ADD Management

In this section, we give a sketch of our previous work on ADD documentation and evolution in a localized software project context.

In order to capture the ADD set, we proposed the Triple View Model (TVM) to clarify the notion of ADDs and to cover key features in an architecting process [5]. The TVM is defined by three views: the element view, the constraint view, and the intent view. This is analogous to Perry/Wolf model's elements, form, and rationale but with expanded content and specific representations [10]. Each view in the TVM is a subset of ADDs, and the three views together constitute an entire ADD set. Specifically, the three views cover three different aspects when creating an architecture, i.e., "what", "how", and "why", as shown in Fig. 1. The three aspects aim to specify design decisions on "what" elements should be selected in an architecture, "how" these elements combine and interact with each other, and "why" a certain decision is made. The detailed contents in each view of the TVM are illustrated in Fig. 2.

Based on the TVM, we define the scenario-based ADD documentation and evolution method (SceMethod) [5]. In the SceMethod, we aim to obtain and specify the element view, constraint view, and intent view through end-user scenarios, which are represented by Message Sequence Charts (MSCs) [11]. Fig. 3 illustrates the SceMethod process. At the beginning of the architectural design process, we obtain initial ADD results. Later on, as the requirements change, the ADDs are evolved and refined according to the newly requirements. By documenting ADDs and evolving these decisions with changing requirements, the SceMethod effectively helps us make architectural knowledge explicit and reduce architectural knowledge evaporation. We have four steps in the SceMethod to derive ADDs in a software project. For the sake of brevity, we will not discuss the detailed steps here. The full illustration of the TVM and the SceMethod can be found in our published work in [6].

IV. ADD Management in Software Ecosystems

In order to manage ADD documentation and evolution in software ecosystems, models and tools should be capable of capturing and representing architectural decisions not only within an organization's architecting process, but also among those external parties in the social community. However, the openness and the sociality of a software ecosystem bring us new challenges and more difficulties of further capturing architectural decisions influenced by a software social community.

We argue that a modified/extended approach to ADDs in software ecosystems should be established, and the corresponding ADD management mechanisms should be extended to fit the ecosystem contexts as well. Note that we propose a paradigm for managing ADDs in open software development environment, which is illustrated in Fig. 4.

In this paradigm, we discuss five aspects of a software ecosystem, i.e., architecture platform, technology, business, community, and resource management. Specifically, a common *architecture platform* is shared by both the organization and the associated third parties to support commonality and stability in the open contexts. *Technology*, *business*, and *community* represent respectively the three types of interactions among the multiple stakeholders. Moreover, a *resource management* mechanism is needed as well to deal with the diverse resources distributed in multiple developments and multiple stakeholders.

As for the ADD management in software ecosystems, we propose a new ADD set that includes basic architecture

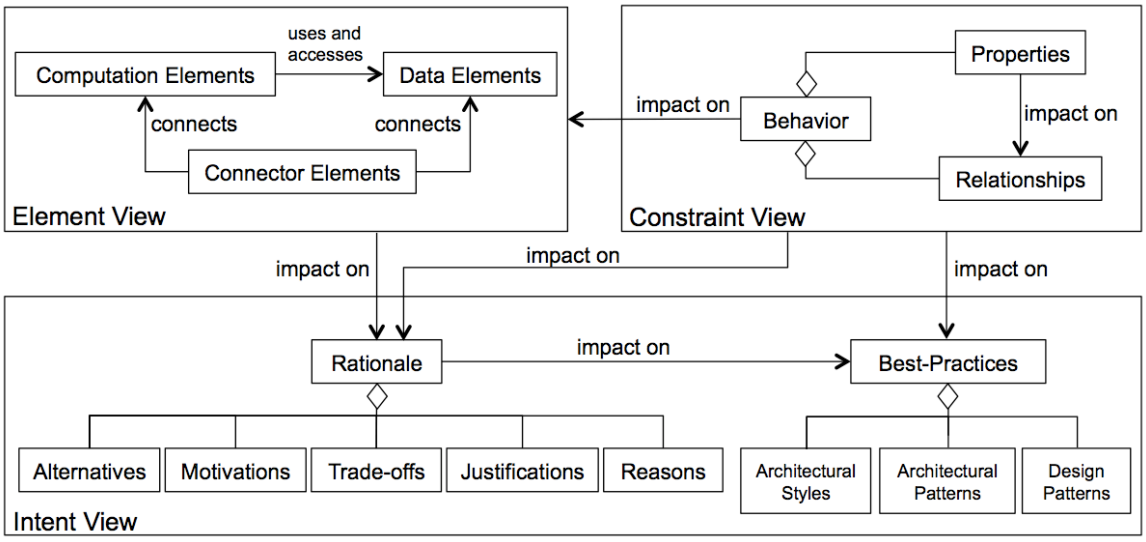

Figure 2. Triple View Model for Architectural Design Decisions

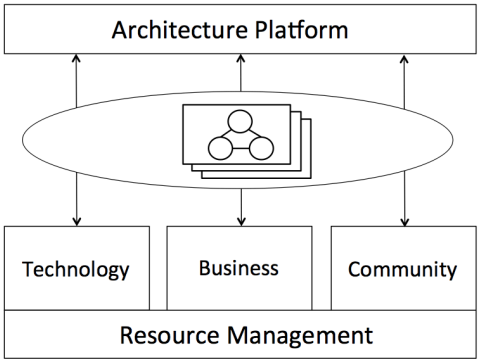

Figure 4. ADD management paradigm for software ecosystems

elements, properties and relationships, and intents that form the common architecture platform, and that extends the decision set by considering decision-making strategies from the technology issues, the business innovations, and the social community. As shown in Fig. 4, we can establish the initial ADD set using the localized model and method, i.e., our TVM and SceMethod that support localized ADD management, while combining the design decisions come from Technology, Business, and Community into the ADD set to constitute a larger set for architectural knowledge needed in software ecosystems.

Note that our paradigm proposed in this paper is still a preliminary ADD framework for the open software development environment, and we believe that many new challenges and difficulties should be considered and be solved as we explore the ADD management in software ecosystems. We present and discuss the expected challenges in the next section, and we aim to offer useful insights into managing architectural knowledge in the context of software ecosystem.

V. OPEN CHALLENGES

We summarize major challenges of developing new technologies and tools for managing ADDs in software ecosystems in this section. In general, how to identify ADDs and how to manage the openness and the sociality of the environment are the main challenges in software ecosystems. Specifically, we elaborate each challenge in the following sections.

A. Comprehensive definition

Aiming to identify and manage architectural knowledge effectively, a definition of what should be considered as ADDs in a software ecosystem is an initial requirement. The existing definition for ADDs in the localized software development are likely not to be sufficient to meet software ecosystem requirements. Thus, we argue that an extended or modified approach to ADD sets for software ecosystems should be established, and that in the new approach ADDs are derived not only from the software organization but also from the third parties.

B. Multi-level communication

Sharing and communicating ADDs within an organization, between an organization and external developers, and among third parties are all inevitable and necessary in a software ecosystem. Therefore, how to keep architectural knowledge consistent in a complex distributed and communicating environment should be addressed. New models and tools of ADD management for software ecosystems should contain mechanisms to deal with the multi-level communication and ensure ADDs to be consistent in the open development environment.

C. Completeness

Our previous work on the TVM and the SceMethod helps to document and evolve ADDs in localized software product development, and they derive and maintain the ADD set as complete as possible. However, for architectural knowledge representation in a software ecosystem setting, an adequate model and tool support are still needed, and an approach to managing the completeness of ADDs in the open development environment needs to be developed as well.

D. Scalability

Since different parties contribute to multiple product developments in a software ecosystem, models and tools for

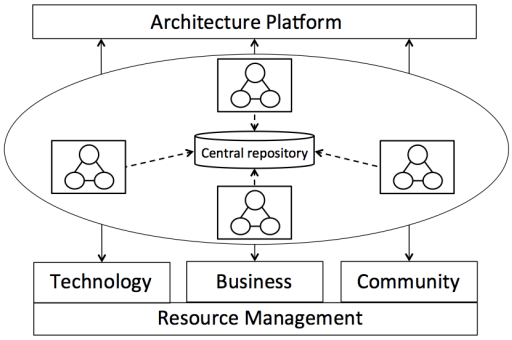

Figure 5. Exploratory solutions for software ecosystems

architectural knowledge management should address scalability issues as the amount and the complexity of architectural decisions increase, and the management on ADDs should also tolerate the continuous involvement of a large array of suppliers, developers and users in an ecosystem. This scalability problem further accelerates the evolution process of ADDs, which is another potential issue in software ecosystem approaches.

E. Traceability

Efficient automated traceability between system drivers (such as requirements, business and market needs) and ADDs is necessary for the enhanced ADD set in software ecosystems. We still need extensive research on architectural knowledge traceability, not only for the localized software development, but also for software ecosystem environment.

VI. Possible solutions

In order to explore the possible directions in research to solve the aforementioned problems and challenges in architectural knowledge management in software ecosystems, we propose several exploratory ideas that can be investigated on our preliminary ADD framework in Fig. 4. The main ideas that we explore are discussed as follows, and we use Fig. 5 to illustrate them briefly.

First, *federated ADD management strategy*. We propose the federated strategy to manage the ADDs in a software ecosystem. Specifically, we conduct ADD management separately focusing on different aspects of the software ecosystem, i.e., we use our TVM and SceMethod to document and evolve ADDs respectively related to the architecture platform, the technology, the business, and the community of the software ecosystem. After we obtain the ADDs from each aspect, we use a central repository to store and record the ADDs, and this therefore suggests the second idea.

Second, *central repository coordination*. The central repository helps us store architectural knowledge information that comes from different areas, and at the same time, it can be accessed by the organization, external developers and third parties, which can be seen as a bridge among different stakeholders in the software ecosystem. Thus, the central repository might be enhanced as a tool to ensure the consistency of ADDs in the software ecosystem contexts. The federated ADD management strategy and the central repository provide good directions for supporting comprehensive definition, multi-level communication, and completeness in software ecosystems.

Third, *configuration management tools*. In order to support the scalability and the traceability in ADD management, one possible solution is that combining ADD management with other configuration management tools. This part of work is expected for both the localize ADD management and the ecosystem ADD management, and we plan to investigate this tool support in our future work.

VII. Conclusions

With the increasing trend of software ecosystems, the management of ADDs becomes more significant and critical due to the characteristics in the software ecosystem context. However, to the best of our knowledge, little work has been conducted in ADD management in a software ecosystem. In this paper, we discuss the typical characteristics of a software ecosystem, and then we briefly present the ADD documentation and evolution method for localized software development. Based on this, we discuss the essential aspects of managing ADDs in software ecosystems, and summarize the expected challenges on ADD management in software ecosystem research and practice. We propose the possible solutions for future investigation.

Acknowledgment

This research is supported in part by NSF CISE Grants IIS-0438967 and CCF-0820251.

References

[1] J. Bosch. Software architecture: The next step. In *European Workshop on Software Architecture*, volume 3047 of *Lecture Notes in Computer Science*, pages 194–199. Springer, 2004.

[2] J. Bosch. Architecture challenges for software ecosystems. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA, pages 93–95, New York, NY, USA, 2010. ACM.

[3] P. R. J. Campbell and F. Ahmed. A three-dimensional view of software ecosystems. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA, pages 81–84, New York, NY, USA, 2010. ACM.

[4] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas. A web-based tool for managing architectural design decisions. *SIGSOFT Softw. Eng. Notes*, 31, September 2006.

[5] M. Che and D. E. Perry. Scenario-based architectural design decisions documentation and evolution. In *Proceedings of the 2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pages 216–225, 2011.

[6] M. Che and D. E. Perry. Managing architectural design decisions documentation and evolution. *International Journal Of Computers*, 6:137–148, 2012.

[7] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool support for architectural decisions. In *Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture*, page 4, 2007.

[8] S. Jansen, A. Finkelstein, and S. Brinkkemper. A sense of community: A research agenda for software ecosystems. In *Proceedings of the 31st International Conference on Software Engineering - Companion Volume*, pages 187 –190, May 2009.

[9] P. Kruchten, P. Lago, and H. V. Vliet. Building up and reasoning about architectural knowledge. In *Quality of Software Architectures*, pages 43–58, 2006.

[10] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17:40–52, October 1992.

[11] D. M. A. Reniers. Message sequence chart: Syntax and semantics. Technical report, Faculty of Mathematics and Computing, 1998.

[12] J. Tyree and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Softw.*, 22:19–27, March 2005.