An Evaluation Study of Architectural Design Decision Paradigms in Global Software Development

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*—Global software development (GSD) is considered as the coordinated activities of software development that are geographically and temporally distributed. The management of architectural knowledge, specifically, architectural design decisions (ADDs), becomes important in GSD due to the geographical, temporal, and cultural challenges in global environments. Based on our previous work on ADD management in localized software development (LSD), we present five ADD paradigms used for GSD projects with different organizational structures. We also investigate the benefits and the challenges of the ADD paradigms by conducting an evaluation of the paradigms using extensive archived semi-structured interview data from industrial GSD projects. We aim to provide a fundamental framework for managing ADD documentation and evolution in GSD, as well as offer useful insights into managing architectural knowledge in a global setting.**

***Keywords*-architectural design decisions; global software development; documentation; evolution**

I. Introduction

Global software development (GSD) is an increasing focus in the field of software engineering. It can be considered as the coordinated activities of software development that are not localized and centralized but geographically and temporally distributed [12]. Little attention has been paid to software architecting processes and software architectural knowledge management in the context of GSD. Similar to localized software projects, software architecting and architectural knowledge are important to support designing, developing, testing, and evolving software. We note, however, that in the global development of large complex systems, architecture plays an even more critical role in the structure of the project [11]. Therefore, managing and coordinating architectural knowledge such as architectural design decisions (ADDs) is a significant and also relatively new research problem in the context of GSD.

In our previous work on ADD management, we had an overall goal of providing a systematic approach that supports ADD documentation and evolution in a localized software development (LSD) context. Based on this, in this paper, we present and discuss five typical ADD management paradigms for global software projects, and we also conduct an evaluation on these paradigms using archived semi-structured interview data from industrial GSD projects to investigate the benefits and the challenges of each paradigm in the GSD contexts. Since little work has been done on ADD documentation and evolution in GSD research and practice, we aim to provide a fundamental framework for managing ADD documentation and evolution in a global setting, and also provide better insights into architectural knowledge management for researchers and practitioners in GSD contexts in the field of software architecture.

To the best of our knowledge, our study is the first to provide ADD management paradigms in GSD projects and to support architectural knowledge management in global settings. Our study provides evidence that *managing ADDs in the GSD contexts reduces the complexity of coordination and integration among multiple distributed sites, decreases misunderstanding among different people, and also offers useful documentation for project planning and other management policies.* Our evaluation is also the first industrial investigation into the benefits and the challenges of global ADD management in practice.

II. Background: ADD management in LSD

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 [3]. 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 [18]. Each view in the TVM is a subset of ADDs, and the three views together constitute an entire ADD set.

Based on the TVM, we proposed the scenario-based ADD documentation and evolution method (SceMethod) [3], and we specified the element view, constraint view, and intent view through end-user scenarios, which are represented by message sequence charts (MSCs) [19]. By documenting all the possible ADDs and evolving these decisions with changing requirements, the SceMethod effectively helps us to make architectural knowledge explicit and to reduce architectural knowledge evaporation. Basically, 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 process of each step. We have the full illustration in [4].

(DOI reference number: 10.18293/SEKE2015-215)

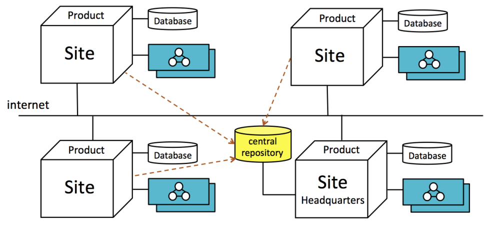

Figure 1. Product-based Paradigm in GSD (for network product)

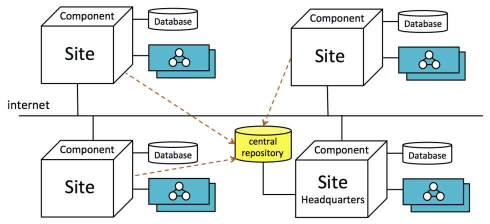

Figure 2. Product-based Paradigm in GSD (for single product)

III. ADD Management Paradigms in GSD

In order to support ADD management in GSD projects, we proposed three strategies for managing ADDs in a distributed context, and discussed how distributed sites coordinate with each other to share and maintain consistent architectural knowledge. The three strategies for multi-site ADD management are federated strategy, client-server strategy, and incremental strategy respectively [5].

Given the foregoing discussion, we develop and discuss the following five paradigms for global software projects. Each paradigm adopts one strategy and is applied to one of the different organizational structures.

1) Product-based Paradigm (Product-based Structure / Federated Strategy): We consider two cases for product-base paradigm, which are shown in Fig. 1 and Fig. 2.

In Fig. 1, the global organization works on a network product (such as the case in our evaluation in the next section), and each individual site is responsible for one individual/dependent product. In Fig. 2, the global organization works on a single product, then the product is decomposed into components and the different components are allocated to distributed sites.

We adopt the federated strategy to manage ADDs in the GSD projects with product-based structures. As shown in Fig. 1 and Fig. 2, each site manages ADD documentation and ADD evolution locally according to the TVM and the SceMethod. In addition, one of the global sites is selected as the headquarters and is to set up a central repository for recording and storing architectural decisions, which enables all the geographically distributed sites to share ADDs in the global context. These multiple sites have the access to the central repository, so that they can check in their local ADDs to the repository, read ADDs come from other sites, and even reuse ADDs from other sites when necessary.

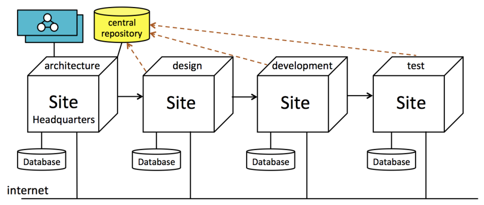

Figure 3. Process-based Paradigm in GSD

The headquarters with the central repository coordinates architectural knowledge in the repository and keep them consistent without conflicts. During the evolutionary process, the evolved ADDs from each site are also transferred to the central repository.

2) Process-based Paradigm (Process-based Structure / Client-Server Strategy): For the process-based structures in GSD, the architecting process mainly occurs in the architecture phase, and all the other subsequent development phases are considered as the clients who access the ADDs derived in the architecture phase. Therefore, the client-server strategy provides us with suitable support for GSD projects with process-based structures.

In Fig. 3, we note that the architecting process is conducted in the site with architecture phase, relying on our TVM and SceMethod to derive the typical ADD set. Moreover, a repository is set up in the same site to manage architectural knowledge documentation and evolution. This repository is also regarded as a central repository among the global sites, and all the other sites have access to the repository for sharing and reusing ADDs in their specific development phases. In some cases, the subsequent development phases, such as design phase, may also come up with new architectural decisions as the process proceeds. However, we do not deal with this kind of exceptions for now, but only explore the general paradigms that are normally used in GSD.

3) Release-based Paradigm (Release-based Structure / Incremental Strategy): The last two paradigms are both for GSD projects with release-based structures. We also discuss two formats in the release-based structures. One is for core-customized releases (which is the case in our evaluation in the next section), and the other is for incremental releases. Since different product releases are allocated to different sites, it is obvious that in the release-based paradigm each site derives its ADD set locally, and maintains ADD documentation and evolution in its local repository.

Figure 4 and Figure 5 show the ADD paradigms for the global projects with release-based structures. In Fig. 4, we can see that the ADDs from the core site are transferred to the customized sites. Besides combining the ADDs from the core site, each customized site has its local ADD management. Similarly, as illustrated in Fig. 5, each repository plays an important role in establishing a bridge to transfer architectural knowledge, which complies with

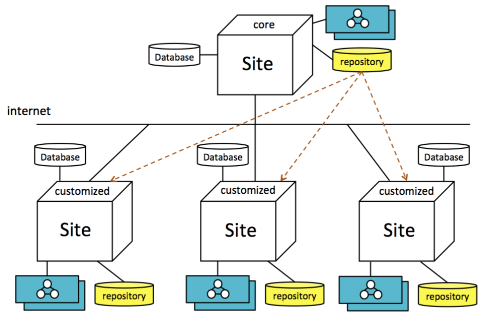

Figure 4. Release-based Paradigm in GSD (for core-customized releases)

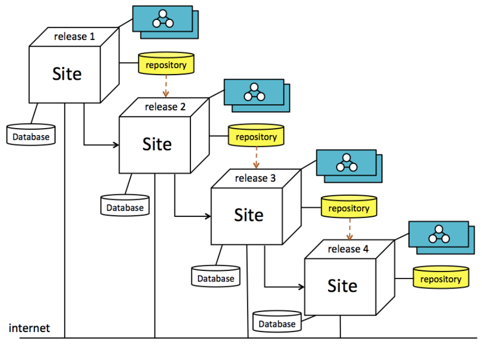

Figure 5. Release-based Paradigm in GSD (for incremental releases)

the mechanism in the incremental strategy. In the release-based structure, the core-customized releases or the multiple releases contain similar or even the same functionalities and product features, which implies that the ADDs derived from these different releases may have similarities as well. By adopting the incremental strategy in these two paradigms, each repository can serve as a reused ADD pool, and it is easy to combine, reuse, and modify ADDs.

IV. Evaluation

In order to compare the ADD paradigms and evaluate whether they will bring benefits or introduce more challenges into global software projects, we investigate the aforementioned paradigms using extensive archived semi-structured interview data from Lucent Technologies [11], a telecommunications systems company with a number of geographically separated software projects. In our evaluation, we discuss the global projects in the following three aspects: degree of autonomy, resource requirement, and coordination complexity, which are the three factors largely influenced by the global settings.

A. Research Questions

We investigate the ADD paradigms by considering the following research questions:
RQ1: What are the benefits of each ADD paradigm regarding the aspects of degree of autonomy, resource requirement, and coordination complexity in different GSD organizational structures?
RQ2: What are the challenges of each ADD paradigm regarding the aspects of degree of autonomy, resource requirement, and coordination complexity in different GSD organizational structures?
RQ3: How do the intent-related decisions and the evolutionary history of ADDs improve architectural knowledge management and project management in GSD projects?

B. Overview of the GSD projects

Twenty-seven interviews were conducted in six different organizations throughout Lucent Technologies. The interviews provided us with information about the project management and project evolution, as well as the distribution of work, and organizational and development situations.

We plan to look into the interview data from four organizations among the total ones. They respectively have different organizational structures. Each organization is briefly described here [11]:
Org_A produces a series of smaller products that are marketed together. Each product is developed by a single site, and all the different sites jointly provide a network product, including a manager component that ensures that all the others work together.
Org_B and *Org_C* build a very large telecommunications product together. They broke up their work into several process steps, and these steps are then used as handoffs among various locations.
Org_D has numerous sites. They produce software that is used for monitoring and managing networks. The basic product is built in USA, and the additional work for deliveries to particular customers is performed in Europe.

C. Analysis

To examine whether the ADD paradigms for GSD projects have visible benefits, or even bring in new challenges, we analyze the interview data from the four organizations above. We identify the characteristics of degree of autonomy, resource requirement, and coordination complexity for each organization, in order to obtain deep insights on the influence of ADD paradigms. We present our analysis in Table I.

As shown in Table I, we can see that in each organization, the interviews are conducted with several different roles in the software projects in order to provide GSD projects information from various points of view. Based on the organizational structures, we respectively adopt different ADD paradigms in each organization. Basically, Table I summarizes the organizations that we investigated from three aspects, which helps us understand how each ADD paradigm works in the corresponding organization.

Org_A has a high degree of autonomy, since each site works on an independent products/components. There are well defined interfaces and component functionalities

Table I
THE ANALYSIS AND COMPARISON OF THE GSD PROJECTS

	Org_A	**Org_B, Org_C**	**Org_D**

Role of Participants	Project Manager;
	Department Head;
	Software Developer;
	Tester;
	Software Architect
		In Org_B:
		Senior Software Developer;
		Technical Manager; Quality Manager;
		Director
		In Org_C:
		Assistant Manager; Development Head;
		Software Developer
			Technical Manager;
			Developer;
			Assistant Architect

Organizational Structure	Product-based Structure	Process-based Structure	Release-based Structure
Paradigm	Product-based Paradigm (for network product)	Process-based Paradigm	Release-based Paradigm (for incremental releases)
Strategy	Federated Strategy	Client-Server Strategy	Incremental Strategy
Degree of Autonomy	Needs network level testing; Has a coordinator for each release; Each site has a very close relation with the element manager; Needs lots of integration testing	The work in one site depends on that in another site; Needs to know the status of each site; Reports issues to other sites; Keeps consistent with requirements; A very clear agreement on handoff policy	A hybrid composition of component separation and process step; Core codes should be done before the customization; Needs to contact with customers; Needs to gather requirement for customization

Resource Requirement	Integration phase needs a lot of people;
	Every site needs experts;
	A defined process;
	Training & Tools
		A well-defined software process;
		Defines the interface between sites;
		A documentation platform;
		Product architecture;
		Experts on each site;
		A stable plan on handoff policy and development process
			Training;
			Documentation system;
			Expertise at custom site;
			Agreed plan for handoff

Coordination Complexity	Coordinates the combination;
	Defines interface across sites;
	Shares documents among sites;
	Phones & Emails & Meetings;
	Travelling
		Phones & Emails & Meetings;
		Travelling;
		Different languages and time zones;
		Web is heavily used
			Coordination with customers;
			Negotiations between customization people and core code people;
			Different languages, cultures, and time zones;
			Phones & Emails;
			Continuous Communication among different sites

Summary	*High* degree of autonomy;
	Normal resource requirement;
	High coordination complexity
		Low degree of autonomy;
		High resource requirement;
		Normal coordination complexity
			Normal degree of autonomy;
			Normal resource requirement;
			High coordination complexity

that contribute to the high autonomy. However, they do need to coordinate features across all the individual products/components and this need of being consistent on features does introduce a high degree of coordination complexity, for they need to coordinate when integrating the products/components to make sure everything works consistently. The following quotes show a few examples of the high autonomy and coordination in Org_A.

"We are doing the testing of the element manager in combination with all the different network elements."
"Make sure the network management can manage the network elements and they also interwork."
"What we will do is try to combine all those products together in a single network."

As for Org_B and Org_C, they have multiple sites with different development phases of the project, and the work in one site depends on that in another site. Thus the main challenges for these two organizations are the high dependency between sites and the agreed handoff plan describing the points on what is to be handed off, and how and when to do so. In our investigation, we found that Org_B and Org_C have low degree of autonomy due to the dependency, as well as high resource requirement especially for a well-defined software process, the interfaces between sites, and a clear handoff plan. The following quotes describe the responses from different interviewees about their work.

"One feature was developed here in X and the other one in Y and I could say that we could not test our feature if we didn't have their feature."
"Well what we needed to know was if the planning that they had in X was just a little bit in front of our planning since that we didn't have to lose time because we just had to wait for them to finish."
"Here I have sort of a pretty well-defined software process."

Org_D has the release-based structure, and it contains one site responsible for the core code, as well as all the other sites for the custom codes. Basically, the customization site obtains the core code from the core site, and customizes the code according to the requirements from local customers. From our investigation on the interview results, Org_D normally requires high coordination, since there are much negotiation between the customization site and core site, as well as coordinations with various customers. We can see more examples from the interview.

"once the allocation has been made of where different processes are going to be developed, there is a need for continuous communication coordination."
"there is much negotiation between customization people and the core code people, but what most of the time happens then is that an expert from our site takes a look at it."

D. Results

In this section, we look into what kinds of benefits and challenges will be brought in when adopting the ADD paradigms in these organizations with different structures, and answer the research questions.

RQ1: What are the benefits of each ADD paradigm regarding the aspects of degree of autonomy, resource requirement, *and coordination complexity in different GSD organizational structures?*

We adopt the product-based paradigm in Org_A. In this paradigm, each site manages its ADDs locally, and the multiple sites share their ADDs in a central repository. This enables us to keep architectural knowledge consistent among different sites, and decreases the resources which are used for product training and documentation. Most importantly, the explicit architectural knowledge provides us with more detailed and clearer architecture issues and specifications, which reduces the complexity of coordination and integration among different sites.

The main benefit provided by the process-based paradigm for Org_B and Org_C is that it is easier to establish a well-defined architecture and software process for the organization with process-based structure. Moreover, the ADDs capture key constraint decisions, which leads to a clear and consistent agreement on the handoff specifications. The recording and sharing of ADDs largely decreases the cost of resource requirement in the global organizations, as well as the issues in the dependency among sites.

Similarly, Org_D with release-based paradigm for the ADD management has high coordination complexity. However, the ADDs can be kept up-to-date and consistent with changing requirements from customers by using our TVM and SceMethod, which provides the core site and the customization sites with consistent project information, and decreases the negotiation between them. Moreover, ADDs offer a agreed plan for handoff policy used between the core site and the customization sites.

Overall, we find that ADD paradigms make ADDs explicit, and the pre-written architectural knowledge decreases misunderstanding in the global development context. In the meanwhile, the communication among different sites is in consistency from the beginning of the project, which reduces the degree of intensive coordination across the multiple sites.

RQ2: What are the challenges of each ADD paradigm regarding the aspects of degree of autonomy, resource requirement, and coordination complexity in different GSD organizational structures?

The main challenge when adopting the ADD paradigms in these global organizations is that more resources are required due to the ADD repositories. For Org_A with the product-based structure, the headquarters site has to set up a repository for storing ADDs, and this would increase the resource requirement but will not influence a lot. We have the similar problem in Org_B and Org_C with the process-based structure, i.e., one ADD repository needs to be set up at the headquarters site. However, for Org_D with the release-based paradigm, each local site needs an ADD repository, which takes up more requirements for hardware and software resources in the entire global project. The other challenge is that the access to ADD repositories among multiple sites also increases the coordination complexity in the global organization.

RQ3: How do the intent-related decisions and the evolutionary history of ADDs improve architectural knowledge management and project management in GSD projects?

In our TVM and SceMethod, we can document the intent-related decisions and also update ADDs when software requirements change. This is consistent when we investigate the GSD project contexts. For the global settings, the ADD paradigms collect the intent from stakeholders located at different sites. During the process of integration and combination, which happens a lot in Org_A and Org_D, the stakeholders are more likely to have consistent architectural knowledge. Therefore, misunderstanding and negotiation among different sites and people are much decreased.

In addition, keeping the evolutionary history of ADDs in the global organizations helps the project maintain the documentation system in GSD settings and reduce the inconsistent issues. Specifically, in Org_D the evolutionary history of ADDs provides us with effective way to track the changes of the requirements, thus providing a better control on the customized requirements from customers, as well as the changing features under each release.

E. Threats to Validity

1) Construct validity: We select to evaluate the benefits and the challenges on introducing the ADD paradigms into the GSD projects. Specifically, we focus on how these ADD paradigms affect the autonomy, the resource, and the coordination of the GSD projects. We believe that what we investigate in our evaluation are commonly used and considered in the ADD management, and thus provide good insights into the research and the practice of architectural knowledge management.

2) Internal validity: The primary threat to the internal validity of our evaluation is overlooking relevant problems in the extensive interview data of the GSD projects. This could affect our analysis on ADD management in global contexts. We controlled for this threat by focusing carefully on the specific organizational structures, i.e., the product-based, process-based, and release-based structures, and narrowing the interview data down to no more than five interviewees.

3) External validity: In our evaluation, we use the archived interview data from software industry to investigate the ADD paradigms. As opposed to formal experiments that generally have an emphasis on controlling variables, our evaluation analyzes the data through observations in an open and unmoderated setting. Our evaluation on the archived data may not generalize to other global projects. We controlled for this threat by observing the three most typical aspects that influence GSD projects, i.e., degree of autonomy, resource requirement, and coordination complexity.

V. Related Work

The key concepts of the traditional view on software architecture are components and connectors [18]. Currently, software architecture is viewed as a set of ADDs [14], [22]. The architectural decisions in the software architecting process are increasingly focused on by researchers and practitioners [10], [16], and ADDs are also considered to be a part of architectural knowledge [17]. In [9], a systematic review for architectural knowledge is presented, and different definitions on architectural knowledge and how they are relevant to each other are discussed as well.

Guidelines for documenting software architecture has been provided in [6], [13], however, those documentation approaches do not explicitly capture ADDs in the architecting process. Recently, many models and tools have been proposed for capturing, managing, and sharing ADDs, most of which are discussed and used within a localized software development context [23], [15] and [21] . A detailed comparison of these existing models and tools has been done in [20]. However, the existing models are hard to support architecture evolution very well [2].

With the increasing attention paid to GSD, ADD management should be able to effectively applied in a GSD setting as well. However, little work has be done on ADD management in the GSD. A few of general architectural knowledge management practices for GSD have been proposed and evaluated in [7] and [8]. Furthermore, a literature review has been done [1] to explore architectural knowledge in a GSD context, and six architectural viewpoints are defined to model GSD systems in [24].

Notably, ADDs have not been widely discussed and supported in GSD, and the aforementioned approaches do not address in detail how to capture, share, and evolve ADDs in a global software project. Our current study in this paper is significantly different from these prior studies by focusing on the ADD management in the global practice and by providing the specific ADD paradigms that can be adopted in the global software industry.

VI. Conclusions and Future Work

In this paper, we develop and discuss five typical ADD management paradigms that can be widely used in GSD, and provide a high-level methodology on how to manage the documentation and the evolution of ADDs in the GSD context. We also investigate the benefits and the challenges of the ADD paradigms by conducting an evaluation on the paradigms using extensive archived semi-structured interview data from industrial GSD projects.

Our study is the first to provide ADD management paradigms in GSD projects and to support architectural knowledge management in global settings. In our future work, we plan to implement the ADD paradigms by providing tool support, and apply them to more GSD projects to investigate their impact on GSD contexts and environment.

References

[1] N. Ali, S. Beecham, and I. Mistrík. Architectural knowledge management in global software development: A review. In *ICGSE*, pages 347–352, 2010.

[2] R. Capilla, F. Nava, and A. Tang. Attributes for characterizing the evolution of architectural design decisions. *Software Evolvability, IEEE International Workshop on*, 0:15–22, 2007.

[3] M. Che and D. E. Perry. Scenario-based architectural design decisions documentation and evolution. In *ECBS*, pages 216–225, 2011.

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

[5] M. Che and D. E. Perry. Exploring architectural design decision management paradigms for global software development. In *SEKE*, pages 8–13, 2013.

[6] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002.

[7] V. Clerc. Towards architectural knowledge management practices for global software development. In *SHARK*, pages 23–28, 2008.

[8] V. Clerc, P. Lago, and H. v. Vliet. The usefulness of architectural knowledge management practices in gsd. In *ICGSE*, pages 73–82, 2009.

[9] R. C. de Boer and R. Farenhorst. In search of 'architectural knowledge'. In *SHARK*, pages 71–78, 2008.

[10] J. C. Dueñas and R. Capilla. The decision view of software architecture. In *European Workshop on Software Architecture*, pages 222–230, 2005.

[11] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: dealing with distance in r&d work. In *GROUP*, pages 306–315, 1999.

[12] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE*, pages 188–198, 2007.

[13] C. Hofmeister, R. Nord, and D. Soni. *Applied software architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[14] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *WICSA*, pages 109–120, 2005.

[15] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool support for architectural decisions. In *WICSA*, pages 4–, 2007.

[16] P. Kruchten, R. Capilla, and J. C. Dueñas. The decision view's role in software architecture practice. *IEEE Softw.*, 26:36–42, March 2009.

[17] P. Kruchten, P. Lago, and H. V. Vliet. Building up and reasoning about architectural knowledge. In *QOSA*, pages 43–58, 2006.

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

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

[20] M. Shahin, P. Liang, and M.-R. Khayyambashi. Architectural design decision: Existing models and tools. In *WICSA/ECSA*, pages 293–296. IEEE, 2009.

[21] A. Tang, Y. Jin, and J. Han. A rationale-based architecture model for design traceability and reasoning. *J. Syst. Softw.*, 80:918–934, June 2007.

[22] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

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

[24] B. M. Yildiz and B. Tekinerdogan. Architectural viewpoints for global software development. In *ICGSE*, pages 9–16, 2011.