A Study of Architectural Decision Practices

Thomas D. LaToza, Evelina Shabani, André van der Hoek
Department of Informatics
University of California, Irvine
Irvine, CA 92697-7440, USA
{tlatoza, shabanie, andre}@uci.edu

***Abstract*—Architectural decisions shape a software architecture and determine its ability to meet its requirements. To better understand architectural decisions in practice, we interviewed developers at two organizations. The results revealed that architectural decisions often become technology decisions, which are in turn influenced by both technical and social factors. Meetings and knowledge repositories help to communicate architectural decisions, but code reviews are ultimately necessary to ensure conformance. Costly changes to architectural decisions are caused by the discovery of an Achilles' heel, an important scenario that cannot be supported by an architectural decision. These findings suggest an important need for social development tools that help developers more easily and successfully share valuable technology knowledge and more effectively make technology choices.**

***Index Terms*—architecture, architectural decisions, empirical study, developer practices**

I. Introduction

Software architectures take shape through architectural decisions that together determine the resulting software architecture and its fitness to purpose. An extensive array of methodologies have been proposed to help developers make architectural decisions [2][5][11], communicate their rationale to team members [10], and share knowledge of architectural patterns [9]. Software architecture researchers conceive of architecting as a scenario driven process, where developers consider scenarios and use a combination of previous experiences, analytical consideration, and intuition to make architectural choices that satisfy these scenarios [7]. A number of methodologies such as the Architectural Tradeoff Analysis Method [1] outline techniques for developers to analyze and compare architectural alternatives for their ability to meet the requirements and needs of stakeholders.

While many methodologies have been proposed, less is known about the social dynamics of architectural decision making in practice. What factors are most influential in the architectural decisions developers make? How do developers communicate architectural decisions to their team? What causes architectural decisions to be revisited? How might the risk of making a poor architectural decision be reduced?

To answer these questions, we conducted a small study in which we interviewed professional software developers at two software companies. In contrast to portrayals of architectural decision making as a green-field endeavor where each decision is considered for its ability to satisfy requirements, we found that architectural decision making was driven by important technology decisions. These decisions, in turn, were influenced by a range of both technical and social factors. Meetings and knowledge repositories played an important role in sharing architectural decisions. But some of the crucial implications of the decisions in practice were communicated and enforced through code reviews, where developers could give tailored feedback to new developers on their understanding of the architecture. Finally, we found examples where developers made architectural decisions that they later chose to revisit. These decisions were caused by an Achilles' heel of the technology that was discovered only in its use, where there was no way for the technology to support an important use case. Building on these findings, we suggest that architectural decision making could be more effectively supported by social development tools that help developers more successfully assess the factors they consider in making technology decisions.

II. Related Work

A large number of processes and methodologies have been proposed for making architectural decisions [2][5][11]. In these methods, developers identify architectural concerns, understand the developmental, operational, and political context in which a system exists, identify architecturally significant requirements, and conduct architectural analysis to define the problem the architecture must solve. Developers generate candidate architectures and evaluate tradeoffs through design knowledge, analysis knowledge of the problem, and realization knowledge gleaned from prototyping.

Studies of software architects have found that they often neglect to adequately document and share their decisions (e.g., [6]). In response, a large body of literature has examined systems and techniques for recording and sharing architectural knowledge within an organization [9]. Approaches include processes, knowledge managements systems, and portals. But the focus is on sharing architectural knowledge *within* an organization. Another approach to share architectural decisions is through catalogs of patterns [4]; books have begun to collect such patterns for specific technologies (e.g., [3]).

III. Method

We conducted a series of semi-structured interviews with 5 developers at a small health information technology company (Site A) and 6 developers at a small telecommunications

startup (Site B). The primary language of both codebases was Java. Participants ranged in experience from recently hired developers to the architect and CTO of the company, with an average of 5 (Site A) and 12 (Site B) years of professional experience. Interviews ranged in length from 26 min to 44 min (average of 33 min) and focused on the architecture of their system, important decisions, revisited decisions, knowledge sharing practices, and code reviews.

IV. Results

A. Making Architectural Decisions

Developers at Site B portrayed architectural decision making as a social process that occurred in meetings between developers. "Everyone gets involved, it's not just one person making the decision." These meetings helped to reach consensus on decisions and communicate decisions to the team. But, at the same time, most of the developers felt the architecture was primarily the product of a single senior developer. Developers at Site A also made decisions socially, but reported meetings as a response to questionable decisions rather than the default practice.

Developers reported that technology decisions were some of the hardest and most important decisions. Technology decisions were first motivated by requirements (e.g., performance, scalability, reliability) and key use cases (e.g., a server in a datacenter crashes). But the decision making process was driven less by finding the perfect match to the desired requirements and more by a range of other factors (Table I). The perceived popularity of a technology played an important role, sending a signal that others believed the technology to be the best solution. Developers also considered if the technology was likely to endure long term, the documentation quality and perceived effort required to learn it, and the experience of operations stakeholders its deployment.

Important technology choices led to key design principles that narrowed possible architecture choices. "I think by choosing something like [Apache] Wicket it kind of enforces a pattern on you." Cognizant of this reality, developers evaluated technologies by judging the architectural styles they might create. Developers preferred technologies that reduced coupling in their system, allowing separate concerns to be encapsulated in separate modules of the system and reducing ripple effects through the system when functionality was added. And developers preferred technologies with APIs that abstracted considerations to which they wished to remain oblivious. For example, developers were very aware of the extra considerations a NoSQL database imposed that a SQL database did not.

Developers preferred technologies that were lightweight – that did not introduce unnecessary complexity from unneeded features – and disliked technologies that were "bloated". Yet, the needed features varied by context. One developer reported that a previous company had chosen JMS for its ability to enable fault tolerance by queuing messages when a service is down; at Site B, it was used for grouping and discarding unnecessary messages.

Different factors may support different decisions, and developers may differ in their perception of each factor. In these cases, developers with seniority may play an important role in adjudicating decisions, with their personal preferences biasing choices. At Site B, several developers felt that an architectural decision was strange, but a more senior developer felt it to be the best choice. Corporate requirements also play an important role – a developer reported that a larger company where they had worked previously had mandated the use of in-house technology.

TABLE I. Factors Developers Reported Considering in Making Technology Choices

Factor	Example
Scalability	"It is easier to scale Tomcat out vertically than JBoss."
Extensibility	"It is easier to plugin open source tools"
Popularity	NoSQL databases are the "hot thing".
Personal bias	Preference to put logic in the database
Corporate bias	Corporate requirement for in-house frameworks to be used
API usability	SQL provides more abstraction than NoSQL, through features such as rollbacks, atomicity, and foreign key constraints
Learnability	Preference for middleware that developers believe they can learn; preference for middleware with clear documentation.
Expected longevity	Preference for technologies that endure
Reduce coupling	JSON allows optional parameters to be added while allowing components that ignore it to be oblivious.
Simplicity	J2EE is "bloated" because much of its functionality is not needed.
Deployment	Operations experience supporting MySQL deployment

B. Communicating Architectural Decisions

Developers used wikis to document major decisions (Site A) and explain the API to end-users (Site B). Developers reported consulting the wiki when making new decisions and looking to understand rationale. But despite these uses, developers felt that only "10% of design decisions and constraints make it to the Wiki, because who has time to write into the wiki." Others felt the key barrier was not a lack of effort or commitment but the rapidity with which the codebase evolved, feeling that the explanations of decisions were complete but outdated. This replicates findings from studies of design rationale [8]. Finally, developers felt that the small size of their organizations made verbal communication particularly important.

Code reviews served an important role in ensuring code's compliance to architectural decisions and communicating these decisions to new developers. Much of the focus of code reviews was on code and design level issues – unclear code, dead code, bugs, naming convention violations, code duplication, not following established patterns, and committing test code. But developers also looked to ensure that

architectural decisions were respected by, for example, ensuring code was implemented using the correct libraries. This was particularly important when developers' past experiences led to code writing habits that conflicted with architectural decisions. One developer reported code that had been implemented in a batch-oriented style – scheduled to run at a fixed time every night – rather than in the project's event driven style. Another developer gave an example of persisting a list of 300,000 items in an http session, which both violated the architectural decision to be as stateless as possible and created a significant performance problem.

C. Revisiting Architectural Decisions

Developers at Site B used an agile, iterative development process. Despite the use of agile, developers still spoke of the value of "design[ing] it right the first time" and the use of prototypes and mockups to avoid mistakes. But upfront design to achieve performance was explicitly discouraged, with a preference to instead optimize for bottlenecks observed in production.

Largely through two complete rewrites of the system, many architectural decisions had been revisited (Table II). Explaining the situation in hindsight, developers described the decision's Achilles' heel: an important use case it could not support. For example, an early decision to rely exclusively on SQL databases would not allow tables to contain billions of rows. Developers viewed the architecture as shaped by learning through experience: "I can tell you that a lot of decisions that we made in the old one were wrong, and the ones that we are making now are much better, but we inevitably will still have mistakes."

TABLE II. TECHNOLOGIES AND PATTERNS DEVELOPERS REPORTED REVISITING

Technology or Pattern	**Achilles' Heel**
J2EE version 1	Entities stored as a database row are stored as a CORBA object, which has much unnecessary data
SQL databases	Cannot scale to billions of rows
Annotation-based AOP	Cannot insert calls in all cases
Unnormalized database	Schema changes require changes to all consumers
In-memory state persistence	When deployment node goes down, state lost

V. LIMITATIONS

Like all studies, our study has important limitations. Most significantly, it was conducted at two sites, and some of the practices observed may be specific to the culture, experience, requirements, and domain of the sites studied. The results are also limited by the reliance on interview data, which was limited by the topics discussed and biased by the recollection of extreme and salient examples. Thus, the results might not be representative of more typical practice. Further work is required to investigate the generality of these findings.

VI. DISCUSSION

Architectural decision-making is traditionally seen as a process wherein developers consider requirements, propose architectural alternatives, and make a decision as to which alternative best satisfies the requirements. Our results found architectural decision-making to be largely shaped by technology decisions. While developers still sought to satisfy requirements, developers found and evaluated technologies rather than architectural decisions. Technologies then imposed important constraints on subsequent architectural decision making. Within these constraints, there was sometimes room for further architectural decisions, but technology decisions were viewed as the most central.

Developers reported revisiting architectural decisions, resulting in expensive architectural changes. Revisiting an architectural decision was driven by an Achilles' heel – a crucial scenario that the architecture needed to support but which the technology or pattern made difficult or impossible. Some of these limitations might be foreseeable – in retrospect, developers viewed them as obvious. If only they had the hard-earned knowledge they gained through experience upfront, they might have decided differently. Moreover, many of the limitations seemed to reflect broad issues, applicable to the technology or pattern's use in a range of contexts. This information, gained at great expense, appears highly valuable to other consumers of a technology or pattern.

While architectural change may be inevitable, especially in an agile process, our results suggest an opportunity for social software development tools to reduce architectural change by helping developers to more effectively share their hard-earned technology experience. Resources such as books, tutorials, forums, and QA sites communicate technology knowledge. But most are focused on making *use* of a technology, not evaluating it for adoption. When such information is available, it is often difficult to find and hard to aggregate, and difficult to compare technologies. A technology's website often helpfully provides a succinct description of its main benefits and selling points. But this information presents the case *for* a technology by its creators and does not feature information about a potential Achilles' heel adopters might encounter. And comparisons with alternative technologies are likely to be biased.

Our results suggest design requirements for a social software development website that helps developers to quickly browse and compare technologies they are considering. The site should be operated by a third party that does not have a vested stake in any of the technologies reviewed. The site should provide and aggregate signals for the factors developers consider when examining a technology (Table I). Such information could be crowdsourced, allowing developers to post about their experiences, with the site aggregating opinions to make judgments. Particularly important to share and highlight are potential Achilles' heels, suggesting a possible "gotcha" an adopter might experience with a technology. Other developers might respond to these with techniques for working around the issue or even debate its importance. And this information could also be valuable to the technology developers themselves, providing valuable feedback on their

technology's perceived important limitations and comparisons against potential competitors.

VII. Conclusions

Architectural decision-making is motivated by requirements but ultimately constrained and intimately connected to technology decisions. Developers today face challenges in making good technology decisions, sometimes resulting in expensive architectural changes. This provides an important opportunity for social software development tools to help more effectively share developers' hard-earned knowledge.

Acknowledgments

We thank the participants of our study. This research was funded in part by NSF grant IIS-1111750. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

References

[1] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice. Pearson, 2003.

[2] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, "Decision-making techniques for software architecture design: a comparative survey." ACM Computing Surveys, 43, 4, Oct 2011.

[3] M. Fowler, Patterns of Enterprise Application Architecture. Pearson, 2003.

[4] N. B. Harrison, P. Avgeriou, and U. Zdun, "Using patterns to capture architectural decisions." IEEE Software, 24, 4 (July 2007), 38-45.

[5] C. Hofmeister, P. Krutchen, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches." Journal of Systems and Software, 80, (2007), 106-126.

[6] J. F. Hoorn, R. Farenhorst, P. Lago, and H. van Vliet, "The lonesome architect." Journal of Systems and Software, 84, 9 (Sept. 2011), 1424-1435.

[7] P. Krutchen, "Mommy, where do software architectures come from?" Proceedings of the Workshop on Architectures for Software Systems (IWASSI), 1995.

[8] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits." Proc. of the International Conference on Software Engineering (ICSE), 2006, 492-501.

[9] R. M. Parizi and A. A. Abdul Ghani, "Architectural knowledge sharing approaches: a survey research." Journal of Theoretical and Applied Information Technology, 4, 12 (2008), 1224-1235.

[10] J. Tyree and A. Ackerman, "Architecture Decisions: Demystifying Architecture," IEEE Softw., vol. 22, no. 19-27, 2005.

[11] L. Xu, D. Richardson, and H. Ziv, "A survey of software architecture decision-making techniques." ISR Technical Report UCI-ISR-07-10, Dec. 2007.