ABSTRACT

The discipline of software engineering is concerned with the development, operation, and maintenance of software [1]. Software engineering research is all about making bigger, better, and faster software. Bigger, as in creating ever more complex and bigger software systems to deal with even more complex problems. Software architecture can be considered as the collection of key decisions concerning the design of the software of a system. Knowledge about this design, i.e. architectural knowledge, is the key for understanding software architecture and thus the software itself. Architectural knowledge is mostly tacit; it only exists in the heads of the creators. A problem is that this type of knowledge is easily lost. This phenomenon is called architectural knowledge vaporization and contributes to a number of problems that the industry is struggling with: expensive system evolution, difficult stakeholder communication, and limited reusability. The central theme of this study and research is how to reduce this vaporization of architectural knowledge. The focus is on one important form of architectural knowledge: architectural design decisions.

KEYWORDS Software Architecture (SA), Architectural knowledge (AK), Architectural Design Decision (ADD).

I. INTRODUCTION

One sub-discipline within software engineering is concerned with studying software architectures, which is a kind of high-level (abstract) design of the software of one or more systems. Currently, there is no agreement to what exactly software architecture entails. This is evident from the hundreds of different definitions found in both literature and the software architecture community [5]. One popular definition is from [6]:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."

Software architecture are created, evolved, and maintained in a complex environment. The architecture business cycle of figure 1.1 illustrates this. On the left hand side, the figure presents different factors that influence software architecture through an architect. It is the responsibility of the architect to manage these factors and take care of the architecture of the system. An important factor is formed by requirements, which come from stakeholders and the developing organization.

Stakeholders Requirements come from many different people and organizations (e.g. end users, developers, project managers, customers, shareholders, upper management, government, maintainers, and sales people), which have an interest in a system.

Developing Organization Besides the organizational goals described in the requirements, software architectures are also influenced by the vision, business strategy, and structure [7] of the developing organization.

Technical environment Standard industry practices and techniques that are commonplace in the architect’s professional community influence software architecture.

Architect’s experience Although positive architectural results in the past are no guarantee for the future, software architects often prefer architecture solutions that have worked for them in the past.

Software Architecture is used for the following purposes:

- **Blue-print-** The major purpose of software architecture is to outline a design.
- **Roadmap-** Software Architecture allows one to plan ahead the evolution of the software of a system and use it as part of a technology roadmap.
- **Communication vehicle-** Software Architecture description can be used as a communication vehicle.
- **Work divider-** Software Architecture can be used as a work divider, as it decomposes software in smaller parts.
- **Quality predictor-** Software Architecture can be used as an early predictor of the quality of a deployed system.

Architectural knowledge:

A recent development in software architecture research is the notion of Architectural Knowledge (AK). AK encompasses the knowledge involved with
Defining architectural knowledge:
As pointed out in the introduction of this section, the notion of AK and what it entails is still subject of ongoing research [14]. This section presents one opinion on this rather broad concept of AK. To get a better grip on this concept, we have developed a meta-model [17] in the Griffin project [24]. This meta-model, or core model as we call it, describes the concept of architectural knowledge. Figure 1.2 presents the four distinct parts, which together make up this concept. In short, these four parts are the following:

![Figure 1.2: The four parts making up the Griffin Core Model [17] of Architectural Knowledge](image)

**Processes** Software architectures influence the processes in an organization and vice-versa [7, 8]. For example, knowledge of the software architecture is instrumental in creating working units for the division of work in an organization [25]. Knowledge about the processes supported by a system is therefore important architectural knowledge.

**People** Many different stakeholders are involved in architecting. Balancing their concerns (e.g. expressed in requirements) and resolving conflicts is an important aspect of architecting. Therefore, knowledge about the people involved, their concerns and relationships is important architectural knowledge.

**Decisions** To come to an architecture design, decisions need to be made. This includes decisions about which of the stakeholders concerns are deemed important enough to be addressed in the architecture design. In addition, it also includes decisions about the architecture design itself, which often require a difficult balancing act between the aforementioned concerns. Knowledge of these decisions is crucial, as they form the basic underpinning of the architecture design.

**Design** Knowledge about the software architecture design forms the cornerstone of architectural knowledge. Central is the notion of the architecture design, which can be expressed in one or more languages (both natural and formal). Using such a language, an architecture design can be captured in one or more artifacts (e.g. word documents, PowerPoint presentations, etc.). Example of these languages to express an architecture design includes Architecture Description Languages (ADLs [26]).

II. ARCHITECTURAL VAPORIZATION
One of the major problems with architectural knowledge is architectural knowledge vaporization. In this process, an organization loses its tacit architectural knowledge. This can happen due to a number of reasons:

- The availability of people for an organization changes over time. For example, employees start working for a competitor or retire.
- Fast changes in a system’s environment, both in business and technology, as it becomes harder and harder to relate the AK to a system’s originally (intended) environment, makes recalling this AK itself difficult.
- Architects consume or produce AK without realizing this fact. Hence, they are unaware of the need to make the AK explicit.
- Making AK explicit is often deferred to a later moment in the life cycle. However, due to the forgetful nature of humans such AK is easily lost.
- The effort it takes to make AK explicit is bigger than the expected benefits. Hence, the organization takes the AK vaporization for granted, i.e. it uses a personalization knowledge management strategy.
- Architects don’t know how to make AK explicit. Losing AK (and thereby architectural decisions) is most critical, as it contributes to a number of problems the software industry is struggling with [33]:
  - Expensive system evolution. Systems need to evolve to keep up with the changing world surrounding it. The requirements a system is expected/required to fulfill change and consequently the system needs to change. Typically, starting from scratch is not an option. Instead, an existing system is often evolved to meet the changed requirements. To evolve a system, new architectural decisions need to be taken. If, however, due to knowledge vaporization, the architectural knowledge is lacking, then adding, removing, or changing architectural decisions becomes highly problematic. Architects may violate, override, or neglect to remove existing decisions, as they might be unaware of...
them. This issue, which is also known as architectural erosion [34, 35, 36], results in high evolution costs.  

• Lack of stakeholder communication. Stakeholders usually come from different backgrounds. If architectural decisions are not shared among the stakeholders, it is difficult to perform tradeoffs, resolve conflicts, and set common goals, as the reasons behind the architecture are not clear to everyone. Knowledge vaporization can lead to architectural decisions not being shared, as an organization becomes no longer aware of all of them.

• Limited reusability. Knowledge vaporization is a direct threat to effectively reusing architectural artifacts. In the first place, an organization needs architectural knowledge to become aware of suitable reuse opportunities. Second, to prevent remaking past mistakes, knowledge is needed of the decision process leading up to the artifact. Third, architectural knowledge is required of the assumptions of these decisions to determine if the artifact is suitable for the situation at hand.

III. RESEARCH METHODOLOGY

The previous section presented the problems architectural knowledge vaporization contributes to. However, a solution was not presented. We can find a (partial) solution to this problem. Following is the research question that can help to find the solution:

How to reduce the vaporization of architectural knowledge?

The concept of architectural knowledge is rather broad. Therefore, this research proposal focuses on a specific part of architectural knowledge. This research proposal therefore concentrates on one of the less investigated types of architectural knowledge; the decision type. Thus, the main research question this research work tries to answer is:

RQ: How to reduce the vaporization of architectural decisions?

The answer to this research question depends on the knowledge management strategy of an organization. Making knowledge explicit is a good solution for organizations using a codification strategy, whereas knowledge redundancy is a good solution for the ones using a personalization strategy. In this research work, the focus is on the codification strategy, i.e. making the knowledge of architectural decisions explicit.

IV. PROPOSED RESEARCH QUESTIONS FOR ARCHITECTURAL VAPORIZATION

The main drawback of the aforesaid research method is the high risk of the artifact failing long before anything is learned about the concept for which proof is sought. In addition, the artifact may become more important to the researcher than the concept that needs to be proved. This is partially due to the research method, which ignores the formulation of a hypothesis up front, but rather lets it emerge during the process.

The proof of concept research method is primarily used to answer the following proposed research questions:

RQ-1: What are architectural design decisions?
RQ-2: How can we model architectural design decisions?
RQ-3: How can we model features in an SPL?
RQ-4: How to deal with architectural design decision dependencies?
RQ-5: How to deal with feature interactions?
RQ-6: How to provide good tool support for architectural decisions?
RQ-7: How can we recover architectural design decisions?

V. CONCLUSION

In the past, the decision part of architectural knowledge did not receive much attention of the software architecture community. The community primarily concentrated on architectural design and architectural processes, leaving out the decision and people aspects.

This paper explores the problem & impact of architectural vaporization in the software architecture and architectural knowledge. Solution of proposed research questions can help to reduce the architectural vaporization.

REFERENCES

13. D. Falessi, G. Cantone, and M. Becker. Documenting decision rationale to improve


