Chapter 8
The GRIFFIN Project: Lessons Learned

Hans van Vliet, Paris Avgeriou, Remco C. de Boer, Viktor Clerc, Rik Farenhorst, Anton Jansen, and Patricia Lago

Abstract GRIFFIN is a joint research project of the VU University Amsterdam and the University of Groningen. The GRIFFIN project develops notations, tools and associated methods to extract, represent and use architectural knowledge that currently is not documented or represented in the system. The research is carried out in a consortium with various industries, both large and small, that provide case studies and give regular feedback. Paraphrasing [327], the goal of the GRIFFIN project can be summarized as "What architects know, and how they know it". In this chapter, we give an overview of the results of the GRIFFIN project, and lessons learned with respect to software architecture knowledge management.

8.1 Introduction

GRIFFIN is a joint research project of the VU University Amsterdam and the University of Groningen, both in the Netherlands. GRIFFIN stands for "GRId For inFormatIoN about architectural knowledge". The project is supported by the Dutch Joint Academic and Commercial Quality Research and Development (Jacquard) program on Software Engineering Research, and runs from 2005–2009. The research is carried out in a consortium with various industrial partners, both large and small. These partners provide us with case studies and give regular feedback. The domains of these case studies range from a family of consumer electronics products to a highly distributed system that collects scientific data from around 15,000 sources to a service-oriented system in a business domain.

Hans van Vliet (✉), Remco C. de Boer, Viktor Clerc, Rik Farenhorst, and Patricia Lago
VU University Amsterdam, The Netherlands, e-mail: [hans,remco,viktor,rik,patricia]@cs.vu.nl

Paris Avgeriou and Anton Jansen
University of Groningen, The Netherlands, e-mail: paris@cs.rug.nl,gradius@fmf.nl

M. Ali Babar et al. (eds.), *Software Architecture Knowledge Management*,
DOI: 10.1007/978-3-642-02374-3_8,

The GRIFFIN project develops notations, tools and associated methods to extract, represent and use architectural knowledge that currently is not documented or represented in the system. In GRIFFIN, Architectural Knowledge is defined as the integrated representation of the software architecture of a software-intensive system) or a family of systems), the architectural design decisions, and the external context/environment.

GRIFFIN was partly inspired by earlier research we carried out in design space analysis. In [57], we analyzed the complete design space of an electronic commerce system. In the end, three feasible solutions remained. For each of these feasible solutions, trade-offs had to be made, and certain requirements had to be relaxed on the way to the final solutions. After the exercise, we confronted an experienced architect in the domain of electronic commerce with our analysis. He told us he knew these three solutions existed. But he also told us he did not (anymore) know of the trade-offs made on the way to these solutions. This architectural knowledge had apparently vaporized. This is typical for many a software development processes. Architectural knowledge is like material floating in a pond. When not touched for a while, it sinks and disappears from sight. The original goal of the GRIFFIN project was to make this architectural knowledge explicit as much as possible, to prevent it from getting out of sight. Paraphrasing [327], this can be summarized as "What architects know, and how they know it". In the next sections, we give an overview of the case studies carried out and the results obtained, and conclude with lessons learned w.r.t.software architecture knowledge management. In terms of the classification given in Chap. 2, our research fits the *decision-centric* view on software architectural knowledge, with specific attention to three of the four trends mentioned in Sect. 2.4: *Sharing architectural knowledge*, *Aligning architecting with requirements engineering*, and *Intelligent support for architecting*.

8.2 The Beginning

In the first year of the project, we tried to characterize the use of architectural knowledge in the Netherlands at that time. We devised a model of architectural knowledge, an abstract conceptualization of the architectural knowledge domain, and applied this model to various participating organizations. We also constructed a series of use cases for architectural knowledge, and performed survey-based research to get insight in the way practitioners view and use architectural knowledge.

8.2.1 Core Model of Architectural Knowledge

Our initial model of architectural knowledge was based on existing literature and our own insights and ideas. We next tried to map actual usage of architectural knowledge

in four participating organizations onto this model. These four organization can be described as follows:

- *RFA*, a large software development organization, responsible for development and maintenance of systems for among others the public sector.
- *VCL*, a large, multi-site consumer electronics organization where embedded software is developed in a distributed setting.
- *RDB*, an SME performing independent software product audits for third parties.
- *PAV*, a large scientific organization that has to deal with software development projects spanning a long time frame (up to a period of more than ten years).

The initial model exhibited a number of mismatches between our theory and industrial practice. The initial model highly conformed to the IEEE-1471 standard for architectural description [155]. IEEE-1471 prescribes the use of so-called 'Viewpoints to describe the architecture from the perspective of different stakeholders. The resulting 'Views (partial descriptions of the architecture) are aggregated in a single architecture description. Although stakeholders and their concerns play a key role in any software architecting process, the tight coupling of the model to IEEE-1471's Views and Viewpoints turned out to be a mismatch with most organizations practice. In hindsight this need not come as a big surprise, since organizations can (and do) use other approaches for documenting their architectures, which need not coincide with the IEEE-1471 way.

From a closer inspection of the mismatching concepts we learned that those concepts could either be expressed in terms of other concepts already present in the model, or as more generic concepts that are used by the organizations. We therefore constructed a new model of architectural knowledge that is both minimalistic and complete. We regard a model as complete if there are no concepts from other approaches that have no counterpart in the model. With 'minimalistic we signify the feature that it should not be possible to express some concepts from the model in any other concepts from the model. Based on these insights we modified the initial model to obtain a model that is both complete and minimalistic. Because of this latter feature, we refer to our model as a core model of architectural knowledge; elements that can be modeled in terms of core elements do not belong to the core.

Our core model of architectural knowledge is depicted in Fig. 8.1. As a result of the minimalistic aspect of this model, the core model leaves room for the use of different architecture description methods, including IEEE-1471. In our core model of architectural knowledge, the concepts of Stakeholder and Concern coincide with the, widely accepted, definitions of these terms in IEEE-1471: a stakeholder is "an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system" [155]. Both IEEE-1471 concepts of Architectural Model and View are subsumed in our notion of Artifact, i.e. an inanimate information bearer such as a document, source code, or a chapter in a book. Storing or describing the Architectural Design in either of these artifacts can be abstracted to a single action to reflect. The Architectural Design can be reflected using different Languages, including models, figures, programming languages, and plain English.

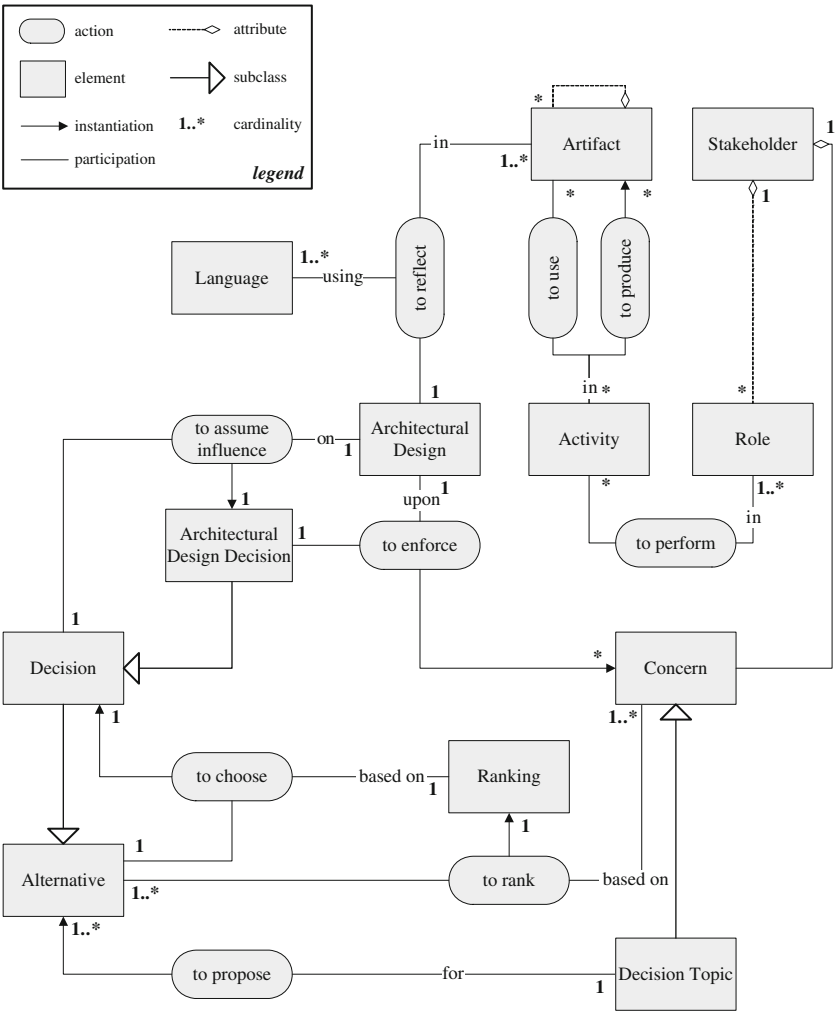

Fig. 8.1 Core model of architectural knowledge

Constructing an architectural design is essentially a decision making process. In our core model, decision making is viewed as proposing and ranking Alternatives, and selecting the alternative that has the highest rank, i.e. the alternative that, after careful consideration based on multiple criteria (i.e. Concerns), is deemed to be the best option available with respect to the other alternatives proposed. It is especially this process of proposing, ranking, and selecting which is hard to articulate and distinguishes the good architects from the weaker. The chosen alternative becomes the Decision. The alternatives that are proposed must address the Decision Topic, and can be ranked according to how well they satisfy this and other concerns.

Architectural Design Decisions are defined as those Decisions that are assumed to influence the Architectural Design and can be enforced upon this Architectural Design, possibly leading to new Concerns that result in a need for taking subsequent decisions. This decision loop captures the relations between subsequent Architectural Design Decisions. This loop also corresponds to the divide and conquer technique of decision making, in which broadly scoped decisions are taken which may result in finer grained concerns related to the broader concern. It also mimics the feedback loop identified in [146].

Although the original model did not entirely fit all organizations, diagnosis of the use of architectural knowledge in those organizations at least showed that each of the organizations has its own perspective on architecture knowledge management, resulting in different issues at each of the organizations. The central issue within RFA was how to share architectural knowledge between stakeholders of a project. The main question within VCL was how compliance with architectural rules can be enforced in this multi-site environment. RDB was mainly concerned with how auditors can discover the architectural knowledge they need to do a proper audit. The main challenge for PAV was how to improve traceability of its architectural knowledge. While the mismatches between theory and practice still prevented us from pinpointing the exact areas of improvement, at least we had an idea where to search for those areas in a next research iteration. These insights provided the basis for the subsequent research in each of these areas, as discussed in Sects. 8.3–8.6. More details about the core model as well as the mapping of architectural knowledge use of the four participating organizations onto this model can be found in [44].

Another use of the core model of architectural knowledge is as a common vocabulary for different organizations. The architectural knowledge of each organization remains expressed in its own terminology, and the core model is used as a shared, reference standard defining the mapping between different knowledge concepts. In our vision, organizations can collaborate on the Web in a grid-like configuration of connected sites, forming a virtual community. In this scenario, the AK mapping via the core model can be used to integrate the services shared on the grid, and therefore facilitate further AK sharing, and collaboration.

8.2.2 The Architect's Mindset

In a next step, we devised a series of typical usages (use cases) of architectural knowledge, and conducted a survey-based study to get insight into the importance practitioners attach to these use cases. These use cases are listed in Table 8.1.

We cluster the use cases based on the purpose of the individual use cases. Some use cases clearly deal with stakeholders only. Consequently, we grouped these use cases into a single cluster. The use case cluster *Architectural decision set* presupposes that a set of knowledge entities (i.e. architectural decisions) and relations between these knowledge are aimed at managing that set. Several other use cases have to do with assessing or reviewing an architecture. Within this *Assessment*

Table 8.1 Use cases for architectural knowledge

Use case cluster	Use cases
Architectural decision set	11. View the change of the architectural decisions over time 15. Recover architectural decisions 20. Identify incompleteness 22. Detect patterns of architectural decision dependencies 23. Check for superfluous architectural decisions 24. Cleanup the architecture
Assessment – reqs.→arch.→impl.	1. Check implementation against architectural decisions 5. Check correctness (i.e. architecture versus requirements) 18. Evaluate the impact of an architectural decision 19. Evaluate consistency 27. Get consequences of an architectural decision
Assessment – risk, tradeoff analysis	4. Perform a review for a specific concern 16. Perform incremental architectural review 17. Assess design maturity 21. Conduct a risk analysis 25. Conduct a trade-off analysis
Stakeholder-centric	2. Identify the subversive stakeholder 3. Identify key arch. decisions for a specific stakeholder 6. Identify affected stakeholders on change 7. Identify unresolved concerns for a specific stakeholder 8. Keep up-to-date 9. Inform affected stakeholders 26. Identify important architectural drivers
Forward Architecting	10. Retrieve an architectural decision 12. Add an architectural decision 13. Remove consequences of a cancelled decision 14. Reuse architectural decisions

cluster, we distinguish between use cases that imply a forward-engineering approach to architecture (i.e. from requirements, to architecture, to implementation), and use cases that target at performing different kinds of analyses and reviews. The first set aims at verification of the architecting activities (are we still on the right track?) whereas the second set aims at validation. Seven use cases form the cluster *Stakeholder-centric*. These use cases concern identification of stakeholders and communication of the architecture to specific stakeholders. The cluster *Forward Architecting*, finally, consists of use cases that create, request, reuse or remove architectural decisions. A summary of the survey results is given in Table 8.2.

The use cases for architectural knowledge within the cluster *Architectural decision set* assume that a set of architectural decisions is at the practitioners disposal. In terms of the use cases, architecting thus boils down to managing and manipulating that set of architectural decisions. Our survey shows that viewing the architecture as a set of architectural decisions and managing that set has not yet transferred to practice, nor is it of particular value to the practitioners.

Table 8.2 Survey results for architectural knowledge use cases

Stakeholder-centric	+
Forward architecting	+
Assessment - reqs.→arch.→impl.	+
Architectural decision set	−
Assessment - risk, trade-off analysis	−

The cluster labeled *Assessment - reqs.→arch.→impl.* covers traceability of architectural decisions to the actual implementation, the relation between decisions themselves, and from architectural decisions back to the requirements that have been set for the information system. Especially respondents in 'construction' roles with respect to architecture (such as architects, designers, developers) regard these use cases as important. This confirms our idea that practitioners involved in the construction of architectures have a need for traceability of architecture. The use cases in the cluster *Assessment - risk, trade-off analysis* are mostly not regarded as important.

A difference that exists between the two subclusters within *Assessment* could lie in the architects mindset. The results of the cluster *Assessment - reqs.→arch.→impl.* reveal a mindset with a linear (i.e. non-iterative) approach to designing an architecture that satisfies the posed requirements and subsequently have the implementation satisfy the architecture. Use cases that offer traceability in this approach are regarded as important. The use cases in the cluster *Assessment - risk, trade-off analysis*, on the other hand, all are aimed at having an intermediate period of reflection to verify what risks apply, or what quality attributes could be affected by certain architectural decisions. These use cases are not directly related to either requirements or implementation. In summary, in contrast to the literature stating that architecture offers a good means to assess the correctness and suitability of the desired solution (e.g. [34, 147]), our results reveal architects regard the use cases for architectural knowledge in the *Assessment - risk, trade-off analysis* cluster as not particularly important. Literature points out that an architecture enables us to assess the design maturity, perform incremental, iterative design reviews, and periodically identify the largest risks pertaining to the architecture. Apparently, these benefits of architecture are not valued by our respondents, which is surprising. Moreover, the use cases in the cluster *Assessment - risk, trade-off analysis* aim at finding possible problems in a certain architecture. Since practitioners do not regard these use cases as important, we might infer that practitioners do not favour a period of reflection in which the current state of the architecture is explicitly tested. Yet, this is one of the main reasons stated in the literature for developing an architecture. Apparently, these intended benefits of architecture have not yet been firmly established in the mindset of architects. The lack of value contributed to the intended benefits reveals a mindset of positiveness ("architects always take the right decisions"), which supports the findings of [314]. Respondents do not like to use architectural knowledge to identify potential weaknesses of their design.

A number of use cases for architectural knowledge are *Stakeholder-centric*. These use cases involve identifying stakeholders and communicating the architecture towards these stakeholders. Five out of the seven use cases in this cluster are regarded as important by the respondents. The remaining use cases identify affected stakeholders on change and identify key architectural decisions for a specific stakeholder are deemed neutral. Furthermore, stakeholder-centric use cases are regarded as more important at the business oriented architecture levels, confirming the general idea that at these levels, communicating architecture to non-IT stakeholders is an important issue. The other way around, practitioners engaged in technical architecture fields do not regard communication of the architecture to stakeholders as important. Apparently, at these levels, practitioners mainly capture architectural decisions for themselves and not for communication to other stakeholders. This in itself is not bad, but reveals that different communication needs exist for different architecture levels.

Four use cases for architectural knowledge fall into the cluster *Forward Architecting*. When we regard the use cases in this cluster we see that adding an architectural decision is deemed important at all architecture levels and by most architectural roles. The use case remove consequences of a cancelled decision is not deemed very important. We can identify two reasons for this. Firstly, this use case requires that a practitioner is able to cancel an architectural decision. Consequently, the practitioner should determine the decision that needs to be cancelled. This requires the practitioner to make a review iteration. Secondly, this use case does not directly contribute to the forward-engineering paradigm we identified when we analysed the *Assessment* use cases. Other use cases in this cluster, such as reuse architectural decisions and retrieve an architectural decision are deemed important by all architectural roles and at all architecture levels. These results show that the practitioners regard architectural decisions as an important asset to be reused in developing a specific architecture.

In summary, the mindset of architects in the Netherlands (as of 2006) reveals an approach which is focused on to create and communicate rather than to review and maintain. This reflects a general pattern as e.g. highlighted in [314]. A more elaborate discussion of our findings is given in [78].

8.3 Sharing Architectural Knowledge

Our research into sharing architectural knowledge took place in a large software development organization, responsible for the development and maintenance of systems for among others the public sector, termed *RFA* in Sect. 8.2.1. We started from the premise that architectural knowledge sharing is best supported by codifying this knowledge in terms of our core model. In our first case study in *RFA*, we found that stakeholders will only share knowledge if the necessary incentives are created [115]. We continued this line of reasoning in a second case study, resulting in a hybrid codification/personalization knowledge management strategy [114], allowing for a

Just-In-Time architectural knowledge sharing portal [113]. In our current work, we are implementing these ideas using Web2.0 technologies such as wikis [116]. This is in line with the trend identified in Sect. 2.4.1 of Chap. 2.

We started our research in **RFA** with a diagnosis of how architectural knowledge was perceived in the organization. We used three main sources for this diagnosis: a questionnaire containing several use cases for architectural knowledge, a documentation study of standards, best practices and architectural descriptions, and finally a set of open interviews with various stakeholders of the architecting process.

As a result, we have distilled four issues related to sharing architectural knowledge in *RFA*:

- *No consistency between architecture and design documents*. There is no alignment between the architecture descriptions and the functional design and technical design documents used by developers and maintainers.
- *Communication overhead between stakeholders*. Developers occasionally have to explain the architects technical decisions more than once. The reason for this is that decisions made in earlier meetings, including the rationale for these decisions, are not adequately stored in the architecture description.
- *No explicit collaboration with maintenance teams*. Although maintainers are targeted in the architectural documentation, they are not involved as a stakeholder in the architecting process.
- *No feedback from developers to architects*. Developers sometimes wear the hat of the architect and also make design decisions. However, architects are not informed on the decisions made by the developers unless explicit meetings take place.

From these observations, we concluded that architectural knowledge sharing was still immature. We next suggested an improved process, essentially centered around better means to codify architectural knowledge [115]. One of our recommendations for example was to establish a central architectural knowledge repository.

Literature though presents warnings that not all knowledge sharing implementations are automatically successful. In [133] several factors that make knowledge sharing difficult are listed, such as the fact that knowledge sharing is time consuming, and that people might not trust the knowledge management system. Another warning is that striving for completeness is infeasible. In addition, we should be aware of the fact that a lot of the available knowledge cannot be made explicit at all, but instead remains tacit in the minds of people [234]. Sharing such tacit knowledge is very hard.

In order to design successful tools for knowledge sharing, a strategy needs to be chosen. In Chap. 1, we distinguished two main knowledge management strategies: codification and personalization. In the architecting process, some architectural knowledge might benefit from a codification strategy, whereas other types of knowledge could be better shared using personalization approaches. A hybrid approach, first coined in [92], is therefore worth considering. Such a hybrid approach could provide a balance between formalized and unstructured knowledge. According

to [141], such a balance is an important prerequisite to stimulate the usage of tools.

To define in more detail how a hybrid architectural knowledge sharing approach should look like we can draw on a study about knowledge sharing by Van den Brink [52]. Van den Brink describes four steps need to be executed in order to create "an interconnected environment supporting communication, collaboration, and information sharing within and among office and non-office work activities; with office systems, groupware, and intranets providing the bonding glue. Firstly, information and explicit knowledge components must be stored online, indexed and mapped, so people can see what is available and can find it (e.g. using digitally stored documents or yellow pages). Secondly, communication among people needs to be supported, by assisting in the use of best practices to guide future behavior and enable sharing of ideas (e.g. emails, bulletin boards, or discussion databases). Thirdly, tacit knowledge needs to be captured using for instance communities of practice, interest groups, or competency centers (e.g. groupware and electronic whiteboards). Lastly, methods are required that offer a virtual space in which a team can collaborate interactively, irrespective of geographic distribution of the team members or time.

We designed and implemented a web-based knowledge sharing portal along these lines: EAGLE – an Environment for Architects to Gain and Leverage Expertise [114]. A second implementation of EAGLE used an Enterprise wiki environment for storing and managing both architectural knowledge and non-architectural knowledge [116]. A screenshot of this wiki is depicted in Fig. 8.2.

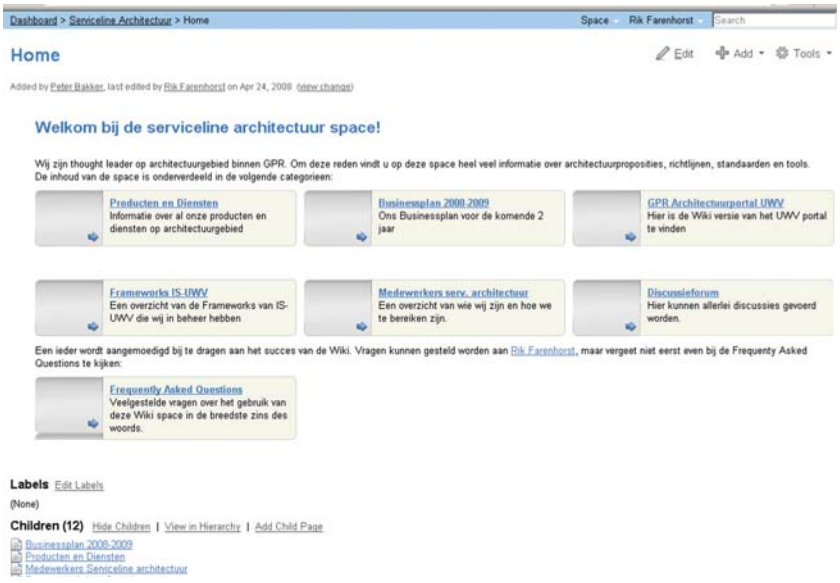

Fig. 8.2 Screenshot of architectural knowledge wiki

To support architectural knowledge codification, parts of the wiki were filled with reusable architectural principles and rules (one per wiki page), after which an overview page was generated using a table of contents plugin. The same approach has been used for codifying architectural knowledge concerned with architectural technologies and patterns. To support architectural knowledge personalization, the wiki offers discussion facilities, the ability to send notifications to subscribed knowledge, and personal pages for all wiki users to mimic a 'yellow pages' system.

By working closely together with **RFA**, over the past few years we have acquired substantial insight in what architects really do, what kind of architectural knowledge needs they have, and how they can be motivated and supported in sharing this knowledge. The tooling discussed above is a first step in making the life of architects a bit easier, but should not be regarded as a silver bullet. In order to further improve the state-of-the-practice, we plan to conduct more empirical research on how architects can best be supported in managing and sharing architectural knowledge.

8.4 Discovering Architectural Knowledge

The research within *RDB*, an SME performing independent software product audits for third parties, started from a very real problem: how to find and comprehend the architectural knowledge that resides in software product documentation [45]. This in turn led to investigating further "ontological" problems: how is architectural knowledge, in particular design decisions, related to evaluation criteria that auditors use when searching for relevant information. We noted a remarkable similarity between architectural decisions and requirements [46].

Auditors have three major questions regarding software product documentation and the architectural knowledge contained in it. These three questions are:

1. Where should I start reading?
2. Which documents should I consult for more information on a particular architectural topic?
3. How should I progress reading? In other words, what is a useful 'route' through the documentation to gain a sufficient level of architectural knowledge?

Auditors who perform a software product audit would greatly benefit from tools and techniques that can direct them to relevant architectural knowledge. We refer to the goal of such tools and techniques as 'Architectural Knowledge Discovery'. A core capability of Architectural Knowledge Discovery is the ability to grasp the semantic structure, or meaning, of the software product documentation. Employing this structure transforms the set of individual texts into a collection that contains architectural knowledge elements and the intrinsic relations between them. A technique we deployed to support the discovery of directions to relevant architectural knowledge is Latent Semantic Analysis (LSA) [205].

In [45], we describe the application of LSA to an example audit case involving 80 documents. The case concerns the reconstruction of the early phase of the software

product audit, in which the auditors need to attain a global understanding of the software product in order to further assess its quality.

In general, when auditors commence a software product audit they want to gain an initial, high-level understanding of the software product. This global understanding is necessary to successfully perform the audit, since it is a prerequisite for subsequent audit activities. For instance, in scenario analyses the supplier of the software product is asked how the product reacts to certain change scenarios or failure scenarios. In order to judge the answer the supplier provides, an auditor needs to have a thorough understanding of the software product. Otherwise, the supplier might provide an answer that is incomplete or inconsistent with the real state of the product, without this being noticed.

Auditors who want to attain overall comprehension of the software product can be guided through the documentation using the semantic structure discovered by LSA. A route that is preferred by all auditors we interviewed is to start with high-level, global information and gradually descend to texts that contain more detailed and fine-grained information. A single term that can be expected to cover the high-level information about the software product well is the term 'architecture'. By using LSA in combination with 'architecture' as the term of interest, we were able to identify a 2-page fact sheet that contained a condensed architectural overview, without ever using the word 'architecture' itself.

Since we were investigating the use and findability of architectural knowledge in quality audits, it was almost inevitable that we ran into the question of the nature of architectural knowledge in this context. A quality audit is different from a 'normal' forward-engineering situation in that auditors will form an opinion about the actual state of the software product and compare that with their opinion on what this state should be. It became obvious to us that in this way auditors take 'virtual' architectural decisions which they compare with the actual architectural decisions taken for the software product.

Since the auditor's virtual architectural decisions form a baseline for comparison, they are often referred to as evaluation criteria. Given their status as things that should be present, one could alternatively refer to them as (architectural) requirements, albeit requirements with a different origin than the client. This, in turn, led to a more general investigation of the relation between requirements and architecture.

Although many would agree that there is some relation between architecture and requirements, the conventional view is that requirements and architecture belong to different domains. From our investigation, we believe that this view may be false. That, metaphorically speaking, architecturally significant requirements and architectural design decisions accumulate in some kind of a 'magic well'. Observers peering into the well see what they wish for. People wishing to find architecturally significant requirements will see architecturally significant requirements; people looking for architectural design decisions will find architectural design decisions.

Currently, the challenges in requirements management and architecture knowledge management are approached as if there are two separate 'information wells'. Both communities perform research on comparable challenges without paying too much attention to what the others are doing. A focus on architectural design

decisions provides an opportunity to recognize and acknowledge that both communities are in fact looking at the same 'magic well' from different angles, we open the door for tighter collaboration between the fields as well as reuse of each other's research results. We see great potential value in further exploring and exploiting the commonalities between architecturally significant requirements and architectural design decisions to enhance support for requirements management and architecture knowledge management alike.

One of the areas in which architecture knowledge management and requirements management meet is in decision support for evaluation criteria selection. We codified *RDB*'s evaluation criteria in a structure derived from the ontology of design decisions of Philippe Kruchten [190], our own work [44], and a collection of informally defined quality criteria within *RDB*. The resulting *quality ontology* can be used to construct an "audit project memory". Such a memory supports reuse of those evaluation criteria in different quality audits. From these codified criteria, new knowledge can be discovered, e.g. through data mining techniques. We are currently implementing a prototype system of an audit project memory that aids authors in their decision making process of including or excluding certain quality criteria in a particular audit.

8.5 Compliance with Architectural Knowledge in Distributed Settings

Nowadays, software development occurs more often in geographically distributed locations. Our research at **VCL** showed that architectural knowledge may be deployed to overcome some of the challenges that are experienced in global software development. Different fragments of architectural knowledge can be shared across different development sites to effect this. A first case study showed that thorough verification of compliance with these fragments is necessary to allow the knowledge to sink in properly at the different development sites [77]. Next, we compared the results obtained at *VCL* with an organization that has adhered to an alternative strategy for communicating architectural knowledge to the development sites [77]. The differences experienced have led us to develop a set of practices to effectively introduce architecture knowledge management in distributed settings [75].

The *VCL* organization is a distributed software development organization for a series of consumer electronics products. The development organization is located at seven sites spread across the globe. Each site has a number of subsystem teams allocated to it that are in charge of developing the subsystem's functionality required for the end products. A central team of architects located at a single site maintains the software architecture and addresses subsystem-exceeding issues like configuration management, subsystem interdependencies, and naming conventions of the various software artifacts. Solutions to these subsystem-exceeding issues were addressed by a set of architectural rules that need to be complied with throughout the organization. The organization, however, found the rules to be formulated

too abstract and did not always understand these rules. We studied the organization to identify reasons for these problems. Our research, described in [77], learned that root cause of these problems was not the description of the rules, but the process by which these rules were developed and subsequently communicated across the development sites. The major improvements included ongoing communication between the central team and the various development sites, implementing compliance verification and explicitly registering deviations from the architectural rules.

To further substantiate these results, we next researched the contribution of architectural knowledge (architectural rules in particular) to overcome the challenges experienced in global software development. We compared *VCL* with another organization (*ABC*) involved in global software development. Whereas *VCL* focuses on the formulation of rules pertaining to the architecture of the product, *ABC* focuses on rules and measures regarding the architecture process. Our research showed that some of the challenges of global software development cannot be addressed by product-based rules alone. In addition, measures in the process are necessary to overcome cultural and team-collaboration challenges experienced in global software development [77].

Reflecting on the research performed and the literature on global software development, we concluded that, based on the differences that exist, there is no one-size-fits-all software development process that addresses all challenges involved in global software development. Consequently, we shifted our focus towards the identification of a set of practices related to the management of architectural knowledge to overcome specific issues. These practices can be incorporated in existing software development methodologies.

We collected a set of practices [75], and characterized these practices by, among others, determining the strategy towards knowledge management supported by these practices (i.e., a personalization strategy or a codification strategy, following [143, 92]). Examples of practices include frequent traveling of key individuals, conducting a shared kick-off meeting in which principles of the software architecture are exchanged, and different forms of frequent communication across development sites. We performed a large-scale validation of these practices at an industrial partner that joined the GRIFFIN project (*VCL2*). This organization has a number of software development centers at different locations and focuses on improving its capabilities by using architecture knowledge management efficiently across these locations. For the validation, we conducted a large empirical study aimed at establishing a baseline of the current architecting practice. As a part of this case study we validated our set of practices for architecture knowledge management with the architects of *VCL2*. We learned, among others, that practices that focus on a personalization strategy for architecture knowledge management are preferred over practices that support a codification strategy.

In future research, we intend to augment these practices by delving architecture knowledge management practices from a series of global software development projects on which independent software products audits have been performed.

8.6 Tracing Architectural Knowledge

The research within **PAV** is concentrated on the traceability between different entities of AK. *PAV* designs and builds large scientific instruments that advance significantly beyond the state of the art. The unique characteristic of this domain, is that these sizeable and complicated systems are often designed during a whole decade before they are actually built. During this long design period, extensive design space exploration takes place where design problems lead to multiple design alternatives with particularly complex trade-offs involved. This exploration takes many different forms: the development and evaluation of (small scale) real world prototypes, quantitative prediction models, and coarse-grained qualitative evaluations. Hence, during this exploration a fairly large amount of AK is created.

In the beginning of the project, **PAV** posed two research questions that concerned the design exploration:

- How can architectural analysis become more transparent? In practice, architecture analysis remained to a large extent a "black art" within *PAV*, since the various analysts possessed very deep and specialized AK that was hardly shared and understood by the rest.
- How are architectural documents related? The sheer volume of documents produced during the design exploration phase rendered the relationships between the document contents practically impossible to locate.

At first glance, providing traceability addresses both these questions. Firstly, traceability should make architectural analysis more transparent by offering an explicit traceable reasoning path between the different analysis models. Secondly, providing traceability both within and between documents makes relationships between documents explicit. This is very much in line with the knowledge management perspective, which requires to make both the knowledge entities and their relationships explicit.

We have come up with an approach towards AK traceability that is based on codifying AK and particularly emphasizes the relationships. The approach, called the Knowledge Architect, consists of a method and a supporting tool suite. The method consists of the following six different activities:

1. *Identify knowledge management issues*. In this activity, we identify the deeper AK management issues an organization is facing. In the case of *PAV*, the need of making the architectural analysis more transparent is derived from the fact that current analysis results are not widely *understandable*. The need of knowing how architectural documents relate comes from the issue that *PAV* is uncertain with respect to the maturity of its design, i.e. is the design complete, consistent, and correct enough to be built and fulfill the requirements?
2. *Derive a domain model*. This activity aims at identifying the actual AK entities that can help in dealing with the issues mentioned in the aforementioned activity. We formalize this knowledge with the help of ontologies [18]. For **PAV**, this means that we identify what AK is important to know and to relate for making a

quantitative analysis transparent (see [164] for an in-depth description), and for inter-relating architecture documents (see [160]). In both cases emphasis is put on making this knowledge traceable.

3. *Capture AK*. Once the AK and its relationships are made explicit, we capture this AK using a codification approach, thereby creating the missing traceability. The supporting tool suite offers several tools that use the ontology of the derived model to assist a stakeholder with capturing relevant AK. To decrease the burden on the stakeholders, these tools are integrated in a non-intrusive way with the tools in which the AK is created and also provide some automated support. For quantitative analysis, we have developed two analysis model tools, one for Python, and one for Excel [164]. We have applied and validated these two tools during architecture analysis at *PAV* concerning performance and cost models. For relating documents, the suite incorporates the Document Knowledge Client, a plug-in for Microsoft Word [160]. Similarly, we have applied this plug-in on software architecture document and validated it within the purpose of architecture evaluation activities.
4. *Use AK*. Once the AK and its relationships have become explicit and therefore traceable, it may be used to address the identified knowledge management issues identified in the first activity. Typically, this involves specialized visualizations to highlight concrete issues. For example, the Document Knowledge Client can colour pieces of text in a document that require further elaboration, when the AK is incomplete, inconsistent, or incorrect. Another example are the dependency graphs rendered by the analysis model tools that allows system analysts in *PAV* to identify the inter-dependencies between the system parameters of their analysis models that influence the design space.
5. *Integrate AK*. Typically, AK is described in various forms and is captured by different tools. To have a complete perspective on the software architecture, the Knowledge Architect uses a central AK repository to integrate the AK of various knowledge sources, e.g. Word, Excel, Python, Excel, etc. Since AK might come from different organizations, each having their own specific domain model, this repository supports ontology mappings. Using these mappings, the AK of different sources can be integrated to form a single overall traceable picture of the architecture. We have developed various models to predict the quality and costs of such mappings (see [206, 207]).
6. *Evolve AK*. AK constantly evolves over the life-time of a system and this evolution also needs to be explicit. The Knowledge Architect tool suite supports the documentation of the AK evolution in two ways. Firstly, the central AK repository offers versioning of the AK in the central AK repository, thereby making the history of AK traceable. Secondly, the repository will be integrated with a SCM to have a traceable co-evolution with the artifacts the AK originates from.

In the future, we want to explore two activities in more depth: evolving and integrating AK. Firstly, we would like to investigate in real life cases *how* the AK evolves over time. Can we perhaps identify patterns in this evolution and strategies to deal with this evolution? Secondly, regarding AK integration, we would like to know to

what extent and under which conditions our predictions of the cost and quality of AK integration hold.

8.7 The GRIFFIN Grid

The core model of architectural knowledge can be used as a common vocabulary for different organizations. Each organization typically develops its own terminology, defined through the years to reflect its business domain, background, know-how and organizational culture. In GRIFFIN, this *local* vocabulary is called *shell* [44].

When different organizations collaborate, it is especially difficult (and time consuming) to understand one another. Imagine to share artifacts and documents expressed according to different terminologies. At the same time, it would be unrealistic for companies to aim at reaching a consensus on one, common terminology, and translate the pre-existing documents accordingly. The GRIFFIN approach is to keep the architectural knowledge expressed in the own terminology of an organization (i.e. according to the shell vocabulary) and refer to a shared standard mapping similar concepts. The core model for architectural knowledge can act as "esperanto" to define the mapping between different shells.

Within GRIFFIN, we envision a virtual and distributed community of collaborating organizations and professionals willing to create and share architectural knowledge. Such a virtual community is meant to support a community of professionals (software architects) to effectively carry out their daily work and further contribute to (and learn from) the community with its own (architectural) knowledge. A combination of strategies for knowledge codification and personalization should provide each individual with the necessary flexibility, to fit in the own working practice and to provide sufficient incentives for successful AK management.

As illustrated in [196], organizations can share AK in a grid-like configuration of connected sites where employees carry out collaborative activities. Software architects can work in their *virtual space* where they can manage their own architectural knowledge and eventually share part of this knowledge with (remote) counterparts in a collaborative social network of professionals.

From a service integration perspective, the core model can be the means to integrate the services that a grid infrastructure may provide [44]. These services may "speak the same language" by exchanging data expressed in concepts from the core model. For instance, the AK codified in the EAGLE portal presented in Sect. 8.3 could be made accessible to other grid partners, and mapped (through the core models) on the AK terms according to their own shell terminology. In a similar way, any of the use cases illustrated in Sect. 8.2.2 can be implemented as a service, and share and exchange data integrated via the core model. In this way, the core model, being shared among multiple sites, realizes a more generic infrastructure.

8.8 Summary

In the GRIFFIN project, we have carried out a number of case studies within participating industries. The close collaboration between research and industry has given us a number of important insights in what works and what doesn't in software architecture knowledge management:

- Different industrial domains have different knowledge models, and we need an infrastructure that deals with that.
- Software architects are not likely to (extensively) codify their knowledge of and by themselves.
- Effective software architecture knowledge management follows a hybrid knowledge management strategy.
- Software architecture knowledge management needs support of lightweight, just-in-time, tools.

It is an illusion to try to coerce different organizations, or even different groups within the same organization, into adopting the same domain structure and terminology. Our core model, as discussed in Sect. 8.2.1 can act as an "esperanto" to allow for a mapping between different local vocabularies. This core model can then be used to share architectural knowledge in a grid-like configuration.

Software architects are busy people. It is their job to find, negotiate, and implement solutions. It is not their job to codify knowledge. At best, we may expect them to partially codify their knowledge. If we deem it important to support software architects in the management of their knowledge, such support of necessity has to be able to deal with incomplete information, such as a design decision without a rationale, or a decision topic with a very incomplete set of design alternatives. To stretch the idea even further, it would be advantageous to be able to mine architectural knowledge from "normal" artifacts produced by architects.

Many knowledge management initiatives start with the idea to codify knowledge. We did too, and learned that this is not the best way to go. Part of the knowledge of software architects will remain tacit, and in circumstances we must be able to find the right person, instead of the right document. Software architecture knowledge management therefore should follow a hybrid strategy, incorporating both codification *and* personalization.

Finally, software architecture knowledge management can successfully be supported by a wide variety of tools. In order to be used at all, these tools should be easy to learn, and fit the daily practice of the architects. If such tools have a steep learning curve, they are not likely to be used. Since architects are already overwhelmed with information, the tools should lead them to the right information at the right time.

Acknowledgements This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research and Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge.