Architectural Knowledge Discovery: Why and How?

Remco C. de Boer

In *ACM SIGSOFT Software Engineering Notes,* Vol. 31, Issue 5 (September 2006)

Architectural Knowledge Discovery Why and How?

Remco C. de Boer
Department of Computer Science
Vrije Universiteit, Amsterdam, the Netherlands
remco@few.vu.nl

ABSTRACT

The need for a method for architectural knowledge discovery stems from the difficulty to find relevant architectural knowledge in the documentation that accompanies a software product. This difficulty arises in particular when the document set is very large, and has been expressed by auditors as a need for a 'reading guide' during a case study we conducted at a company that performs software product audits. Based on the needs of these auditors, we identify the main characteristics an architectural knowledge discovery method should exhibit. This paper argues that Latent Semantic Analysis (LSA) is a promising technique for architectural knowledge discovery.

Keywords

Architectural knowledge, knowledge discovery, knowledge management, software architecture

1. INTRODUCTION

When shared by means of written documentation, architectural knowledge is often not confined to a single document. Instead, the architectural knowledge of a single, large software intensive system is usually distributed over many different documents. Each of these documents may target different stakeholders. For instance, a project manager might be particularly interested in the overall picture contained in a high-level document, whereas a development team needs to learn the effect of architectural decisions on the implementation from the technical specification. This means that each document has a different focus, and between the documents the levels of abstraction used may vary wildly. Furthermore, various explicit and implicit relations exist between the topics and concepts described in the documents, as well as between the documents themselves.

In the case of a software product audit, auditors will typically first use the architectural knowledge contained in the documents to familiarize themselves with the system. Subsequently, the documentation is used as a source of evidence for findings about the software product quality. However, for large sets of documents it can be very hard to locate the knowledge needed for a specific purpose.

How to find the architectural knowledge needed is in fact the most important issue we encountered during a recent case study. In this study, we investigated the role of architectural knowledge in software product quality audits. Architectural knowledge needed by the auditors to assess software product quality comprises architectural design decisions [1] and their rationale, as well as the architectural design of the system [4] and the relations the system has with its environment. Auditors performing these audits have to deal with large quantities of documents that contain such architectural knowledge. The auditors indicate they are in need of a reading guide that helps them discover the relevant architectural knowledge in the documentation. However, such a reading guide is often hard to obtain. We therefore need a mechanism to support (semi-)automated discovery of the important architectural knowledge in a set of documents.

This paper discusses what an automated architectural knowledge discovery method should look like. What are its main characteristics and how do these aid the search for important architectural knowledge in large document sets? Furthermore, we identify a technique that supports attaining automated architectural knowledge discovery. This technique, latent semantic analysis, has already proven itself in other application areas, including the discovery of the structure of source code components [6–8].

2. ARCHITECTURAL KNOWLEDGE USE IN SOFTWARE PRODUCT AUDITS

ISO/IEC 14598 [3] defines a software product as 'the set of computer programs, procedures, and possibly associated documentation and data'. Quality is defined as 'the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs', whereas quality evaluation is a 'systematic examination of the extent to which an entity is capable of fulfilling specified requirements'. A software product audit, in which the quality of the software product is evaluated, can then be defined as

> The systematic examination of the extent to which a set of computer programs, procedures, and possibly associated documentation and data are capable of fulfilling specified requirements.

We conducted a case study at a company that has been performing independent audits of many different software

SHARK'06 June 11, 2006, Torino, Italy.

products. Their audit process is based on the process outlined in ISO/IEC 14598. Customers who request such audits range from large private companies to governmental institutions. Over the years, the company and its auditors have built up a broad experience with software product audits.

In the case study, we addressed the use of architectural knowledge in a software product audit. We were particularly interested in how the auditors gain the architectural knowledge needed to answer the audit question. During the case study we observed an audit the company conducted for one of its customers. Part of this observation consisted of attendance of the audit team meetings. Furthermore, we held interviews with five employees that had all been involved in audits, two of whom were directly involved in the observed audit. The interviewed employees have varying levels of experience and different focal points in conducting audits.

Particular methods that are used to gain architectural knowledge are source code analysis, scenario analysis, interviews, and document inspection. Both scenario analysis and interviews require extensive interaction with the supplier of the software product. Source code analysis and document inspection can, however, be performed using only the source code and documentation that has been provided to the auditors. This renders the latter two techniques important vehicles for any software product audit.

While document inspection is an important part of a software product audit, the sheer amount of documentation that accompanies most software products makes it a difficult method to use. Auditors are often swamped with documents and do not have immediate insight into the overall structure of the document set and the architectural knowledge it contains. This insight is of paramount importance for the audit process. For instance, in scenario analyses the supplier is asked how the product reacts to certain change or failure scenarios. Without a thorough understanding of the architecture of the software system, the auditor is unable to judge the supplier's answer. In that case, the supplier might provide an answer that is incomplete or inconsistent with the real state of the system, without this being noticed. The auditor also needs to distill evidence from the documents that supports his findings regarding the quality of the software product. An overall understanding of the software product is necessary to ease the search for this evidence.

The auditors indicate that, to gain insight into the architectural knowledge, they are in dire need of a 'reading guide' for the document set they receive as part of the product. Such a reading guide tells the auditors where to start reading, and which documents to consult when in need for an answer about a particular topic. Unfortunately, the auditors practically never encounter a reading guide among the documentation of a software product. This means that they have to fall back on interviews with for instance architects to introduce the architecture and documentation. These interviews are not always possible, for example due to time constraints.

The issue the auditors from our case study encounter can be extended to a more general public. All stakeholders that need to familiarize themselves with a software product for whatever reason will have to answer the same question: where in the documentation do I find the architectural knowledge I need? This is especially true when, unlike for some software product audits, no one is available to introduce the architecture to the stakeholder. The stakeholder then has to rely solely on the documentation that accompanies the product. A particular situation in which this can be the case is when a software product has to undergo maintenance. If the original architects are no longer available, the maintenance team only has the documentation at its disposal. We argue that a solution to this problem of insight lies in architectural knowledge discovery.

3. ARCHITECTURAL KNOWLEDGE DISCOVERY REQUIREMENTS

Based on the observations above, this section describes what characteristics an architectural knowledge discovery method should exhibit. The requirements of such a method are aptly summarized by one of the auditors in the following quote:

> *"Tell me where I should start reading, which documents I can consult for more detail about, let's say, functionality or about the architecture, or [for traces] from architecture to design. Provide me with a route through the documentation to bring me up to a certain level of knowledge."*

This was the answer of an auditor when asked what the ideal reading guide should provide. An architectural knowledge discovery method that solves the problems the auditors encounter should bear such characteristics that the requirements posed in this quote are met. In other words, it should 1) tell where to start reading, 2) tell which documents to consult for more detail on a topic, and 3) provide a route through the documentation.

Since software product documentation consists for the largest part of natural language text, a technique that aims to discover knowledge contained in the documentation should be able to 'understand' natural language. A product document set typically contains information on many different topics, including high-level system architecture, functional design, logical design, and infrastructure architecture. These topics are not confined to a single document, but have relations with other topics in other documents as well. To gain a global understanding of the software product, these relationships need to be grasped by the auditor. In other words, the semantic structure of the information in the documents needs to be identified.

Discovering the semantic structure of the document set forms the core of architectural knowledge discovery. Grasping this structure transforms a set of individual texts into a collection that contains architectural knowledge elements and the intrinsic relations between them. This transformation is the foundation for further analysis outlined below.

1) Where to start reading

The semantic structure of a document set can be seen as a measure for similarity; documents that contain semantically related concepts are more similar to each other than documents that describe unrelated concepts. An architectural knowledge discovery method should exploit this intra-document similarity to group related documents together.

Documents grouped together in a cluster have certain semantic correspondence. For the auditor, this can be interpreted as an overview of which documents 'belong together'. If the topics that define the clusters are presented to the auditors, they can select documents that center around high-

level concepts and read those before documents that describe concepts of a lower level of abstraction. This aids the auditor in deciding with which documents to start reading.

2) Which documents to consult

The question which documents to consult for more detail on a topic is an information retrieval question. Successful knowledge discovery should improve the success rate of information retrieval, not only by providing the auditor with insight that can be used to formulate better queries, but also by allowing the discovered semantic structure to be used in the retrieval. By taking the semantic structure into account, documents that contain text semantically similar to the query posed could be retrieved in addition to documents that contain the query terms themselves. In this sense, architectural knowledge discovery is complementary to information retrieval.

3) A route through the documentation

Architectural knowledge discovery should also employ the semantic structure to identify relationships between the different concepts themselves. Together, concepts and relationships provide a 'map' of the architectural knowledge contained in the document set. This map crosses the boundaries of individual documents, linking related topics that are not described in a single document. For example, one document may describe how certain decisions affect some of the product's quality attributes, while another document contains information on the effects of these decisions on the architectural structure of the system.

An architectural knowledge map can be used by the auditor to gain a further understanding of the architectural knowledge in the documents. The map should combine product specific concepts - such as subsystems, use cases, functional requirements, and the like - with more generic concepts, including quality attributes, architectural patterns, etc. Such a combination simplifies understanding the product, since unknown product specific terminology is linked to an idiom well-known to the auditor. A prerequisite for the success of such a combination is that the generic concepts, such as 'service oriented architecture' or 'reusability', are also contained in the documentation and are linked there to product specific concepts, such as 'subsystem X', for instance with the sentence 'To improve reusability, subsystem X implements a service oriented architecture'. The architectural knowledge map provides a route through the software product documentation by conveying which concepts from the documentation are related.

4. LATENT SEMANTIC ANALYSIS

Inspired by the work of Maletic et al. [6–8], we identify Latent Semantic Analysis (LSA) as a technique that supports the goals of architectural knowledge discovery. LSA is a technique that infers the meaning of words and passages from natural text. It does so by statistical analysis of the context in which words are used [2].

Maletic applies Latent Semantic Indexing (LSI, an alias for LSA in the information retrieval field) to recover traceability links between external documentation and source code. An important difference with architectural knowledge discovery is that the latter does not focus on software components, but rather on the documentation that accompanies a software product.

LSA starts with building up a matrix of word-context occurrences. Context can be defined as a single document, or an arbitrary smaller unit such as a section or paragraph. The matrix elements consist of a weighted frequency count, indicating the number of times the word occurs in the context. This results in a high-dimensional sparse matrix. The dimensionality of the represented vector space is reduced by application of singular value decomposition. The resulting matrix is an n-dimensional reconstruction of the original matrix, with n being the reduced number of dimensions. For a detailed example of this mechanism please refer to [5].

The n-dimensional reconstruction is an approximation of the original matrix. Because of the dimensionality reduction, the new word-context estimations represented by the cells in the matrix differ sometimes significantly from the original values. The interesting characteristic of this new matrix is that it no longer only links words to contexts in which they are present, but also words that are somehow related to this context. It has been argued that LSA has the ability to model human conceptual knowledge [5].

From this description it is clear that LSA is able to handle natural language text. Moreover, LSA does not need to have any background knowledge at its disposal. The technique works on statistical analysis alone. This is important, because architectural knowledge discovery will often need to be applied to a largely unfamiliar domain.

The (latent) semantic structure that is discovered with LSA can be further analyzed as outlined in the previous section. Document clustering can for instance be performed by taking 'document' as the measure of context in the word-context matrix. The documents are then represented as vectors in the vector space generated by LSA. These vectors correspond to the columns in the reconstructed matrix. Document similarity can therefore be calculated by calculating the vector similarity. The documents that describe similar concepts will have similar document vectors.

Whereas the columns in the LSA matrix are representations of the documents (or contexts), the rows represent concepts (or words). The similarity between concepts can be calculated analogous to the similarity between documents, using the matrix rows as concept vectors.

Note that the concept of vector similarity can also be used for information retrieval purposes. The query is then translated to a vector and compared to the document vectors in the vector space. Because the document vectors result from LSA, a query can also retrieve documents that are related to the query, but do not contain the query terms itself. This evades the problem of synonyms in the document set, where searching for 'reliability' would not retrieve documents that contain information on 'fault tolerance'.

5. PROOF OF CONCEPT

To demonstrate the applicability of LSA to architectural knowledge discovery, we applied LSA to a small collection of texts. Because we used a small set of texts, we were able to obtain a proof of concept without having to automate the complete analysis. The word-context matrix could for instance be easily constructed by hand.

The texts we used for the proof of concept were taken from three documents that belong to a real software product. This software product was subject to an audit we observed as part of the case study described in Section 2. The audited software product has a service oriented architecture (SOA).

The three documents are at different levels of abstraction; document A contains a detailed description of a single use case, document B contains the functional specification of one of the services, and document C contains a high-level description of the software architecture.

From each of the documents we took a small but representative section. From the use case document we took the introduction of the use case. From the service specification we took the description of the part of the service that relates to the selected use case. From the architecture description we took the section that describes the conformance to SOA. For these three texts, the word-context occurrences were determined, not counting stop words. This resulted in a matrix containing some 90 distinct words used in the three texts. Table 1 shows part of this matrix. Note that the words 'domain entity', 'use case', and 'business object' have been substituted for the product-specific terminology.

Table 1: Word-Context Occurrences

	document A	*document B*	*document C*
domain entity	0	2	0
service	0	3	1
SOA	0	0	3
system	0	0	4
use case	1	0	0
business object	8	3	0

Using LSA, we calculated a two-dimensional reconstruction of the original word-context matrix. Table 2 depicts the resulting figures for the same selection of words shown in Table 1.

Table 2: LSA Result

	document A	*document B*	*document C*
domain entity	0.905	0.827	0.39
service	1.207	1.436	1.52
SOA	-0.451	0.585	2.806
system	-0.601	0.779	3.741
use case	0.651	0.452	-0.15
business object	6.566	4.859	-0.618

After the application of latent semantic analysis, some interesting patterns appear regarding the semantic structure of the texts. For instance, the original matrix shows three occurrences of the word 'SOA' in the text from the architecture description, whereas the other 2 texts do not contain this word at all. After LSA has been applied, a slightly positive association appears between the concept 'SOA' and the text from the service document. On the other hand, the word 'SOA' is inferred to have a negative association with the use case document. Overall, high-level concepts in Table 2 have high association with the high-level architecture description, some association with the service description, and a negative association with the low-level use case document. The inverse is true for low-level concepts such as 'use case' and 'business object'. Another interesting result is the nearly identical association of 'service' with each of the documents (Table 2). This contrasts with the distribution of occurrences of this word over the three texts (Table 1). This makes sense, because in this software product a 'service' is both an element of the service oriented architecture and the means to implement a use case.

6. CONCLUSION

The case study we conducted at a company that performs software product audits indicates there is a definite need for architectural knowledge discovery. Architectural knowledge discovery should provide insight into architectural knowledge that is distributed over various documents. This is not only relevant for auditors, but for all stakeholders that need to familiarize themselves with a software product.

We believe that Latent Semantic Analysis (LSA) is a promising technique for architectural knowledge discovery. LSA infers the meaning of words and the contexts in which these words appear by statistical analysis. In doing so, LSA discovers the latent semantic structure present in a collection of documents. This semantic structure can be employed to satisfy the requirements for successful architectural knowledge discovery: tell where to start reading, tell which documents to consult, and provide a route through the documentation.

A proof of concept provided in this paper shows promising results for the applicability of LSA to architectural knowledge discovery. The proof of concept shows that, for a small sample of texts taken from real software product documentation, LSA is capable of correctly inferring the semantic structure of these texts.

Acknowledgement

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

7. REFERENCES

[1] J. Bosch. Software architecture: The next step. In *Software Architecture: First European Workshop (EWSA)*, 2004.

[2] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science (JASIS)*, 41(6):391–407, 1990.

[3] ISO/IEC. Information technology - software product evaluation - part 1: General overview. Technical Report ISO/IEC 14598-1, 1999.

[4] P. Kruchten, P. Lago, H. v. Vliet, and T. Wolf. Building up and exploiting architectural knowledge, 2005.

[5] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

[6] J. I. Maletic and N. Valluri. Automatic software clustering via latent semantic analysis. In *14th IEEE international conference on Automated Software Engineering (ASE'99)*, 1999.

[7] A. Marcus and J. I. Maletic. Recovering documentation to source code traceability links using latent semantic indexing. In *ICSE 2003*, pages 125–135, 2003.

[8] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic:. An information retrieval approach to concept location in source code. In *11th Working Conference on Reverse Engineering (WCRE 2004)*.