2011 Ninth Working IEEE/IFIP Conference on Software Architecture

Software Architecture Documentation: The Road Ahead

Antony Tang
Department of Computer Science
VU University Amsterdam
The Netherlands
atang@cs.vu.nl

Peng Liang
State Key Lab of Software Engineering, Wuhan University
Wuhan, China
liangp@sklse.org

Hans van Vliet
Department of Computer Science
VU University Amsterdam
The Netherlands
hans@cs.vu.nl

***Abstract* - The basic format in which software requirements and architecture designs are documented is essentially file-based, and it has persisted for decades. Current indexing methods used in file-based documentation are not conducive to retrieving software knowledge. We propose to index software documents with a suitable lightweight ontology to improve the retrieval and traceability of software knowledge. Initial results from a prototype implementation have shown promising prospects.**

Keywords: Software Architecture Documentation, Knowledge Sharing, Ontology, Semantic Wiki

I. INTRODUCTION

The basic format in which software requirements and architecture designs are documented has not changed in many decades. It is essentially file-based with the support of semi-automated tools such as RequisitePro and DOORS for managing requirements, and UML modeling tools such as Rational and Enterprise Architect for documenting design structures[1]. There are two main issues. Firstly, except for a table of contents, there are typically limited capabilities for cross referencing information, and important requirement and design details are difficult to find. Therefore, file-based specification for communicating system requirements and design has its limitations. Secondly, the editing and updating of software documentation such as requirements and design requires coordination on sharing files, this approach is not well-suited to capturing information effectively in a large and distributed team. It deters analysts and designers from keeping documentation up-to-date. As such, we challenge the effectiveness of this traditional software documentation paradigm. In this position paper, we suggest a new approach to improve the capturing and retrieval of software knowledge.

In a case study, we have observed that requirements engineers, architects, developers, and testers who updated their specifications in a file format are unclear about their documentation responsibility. The ambiguity of documentation responsibility and the clumsiness in file sharing prevent software workers from updating formal specifications after their initial creation [1]. Instead, much of the information is documented in technical notes, personal notes, emails, minutes, and wikis etc. and this information cannot be shared effectively. As a result, a large part of the software knowledge is scattered, incoherent, and incomplete.

The retrieval of documented information for subsequent use as knowledge is ineffective. Firstly, requirements and designs continue to evolve. As formal specifications are not updated regularly and completely, people become less reliant on them and they can quickly become out-of-date with the latest development. Secondly, design information is often scattered over a number of specifications and cross references between these specifications are often missing. Keyword search is often used as a means to find related information. However, such searches may not yield accurate knowledge retrieval results because of versioning issues and the use of synonyms and homonyms. Thirdly, many inter-related requirements, designs, and viewpoints exist in large and complex software systems. For instance, a change to one requirement may create rippling effects to other requirements and designs, making it more difficult to keep the requirements and the designs consistent. Furthermore, the relationships between requirements and design are often implicit and hidden. Although many traceability methods and tools exist [2, 3], they remain largely specialized research tools and lack the flexibility for commercial use.

The problems of producing and consuming software documentation can also be attributed to the general lack of understanding of how to create and use documented software knowledge. The software industry and the community have taken the current practice for granted. In this research, we study this subject from two perspectives. Firstly, we investigate how software knowledge is captured, communicated, and used. Secondly, we investigate the use of ontology and semantic annotation to help document and search for software knowledge. This paper describes our approach, some initial results and key research challenges.

II. DOCUMENTATION PRODUCTION AND CONSUMPTION

Large-scale software design and development is a team effort and the timely sharing of software knowledge is important. Many software development organizations use files to share knowledge as captured in requirements and architecture design documents. A file-based documentation method has a number of drawbacks: (a) a file is not easily shareable in editing despite configuration management; (b) the update and release of a file to interested readers is periodic and untimely, especially when requirements and designs are evolving rapidly, so communicating important information can be delayed; (c) obtaining knowledge through a table-of-content or keyword search is ineffective.

[1] This research focuses on the common practices in the software industry. Advanced methods such as ADLs are outside the scope of this work.

978-0-7695-4351-2/11 $26.00 © 2011 IEEE
DOI 10.1109/WICSA.2011.40

These issues occur because of how we currently produce and consume software documentation.

A software architect, as a knowledge producer, would contribute knowledge such as design decisions to other stakeholders who might need to know. The timeliness of such communication is lost if one relies on the release process of a formal specification. Furthermore, decisions may not be communicated to all concerned parties, and the knowledge during the communication may be lost if it is documented unsystematically, say using emails. This issue is exacerbated when there are many knowledge producers in a software development organization, perhaps due to agile development process, multi-site software development, and generally increased system complexity and more technical specializations.

There are many roles in software development, and they rely on the information created by others to make decisions on e.g., requirements and designs. For instance, a software architect needs requirements from business analysts, technical design from a system architect, project information from a project manager, design constraints from software team leaders, and so on. Since documented knowledge is often incomplete and out-of-date, tacit knowledge using personal communication is heavily used [4], but this is ineffective and error prone because knowledge availability is conditional upon the source of the knowledge. Knowledge is available only when the person who possesses the knowledge can be identified, and s/he is willing to share that knowledge, and s/he communicates the knowledge clearly and completely.

If documented knowledge is meant for communicating requirements and design ideas during the development and maintenance of a system, effective search mechanisms to help retrieve knowledge become an important consideration. Currently, file-based documentation typically relies on indexing mechanisms such as the table of contents, and supported by mechanisms such as a keyword finder, traceability matrix, or requirements management tools such as RequisitePro. Additionally, modeling tools provide some sort of structure to relate and retrieve knowledge. For instance, UML tools typically support information organization by diagram grouping, viewpoints, hierarchical sub-system structures, and keyword search facilities. These methods provide limited capabilities for software workers to find the right information.

Recent studies in architecture knowledge sharing have produced a number of meta-models and use cases [5] but little of the recent works address what and how software workers can improve the effectiveness of retrieving and relating documented knowledge. Software knowledge is about providing understandability and relevant information of software artifacts, supporting traceability and impact analysis, and it must be kept up-to-date and trustworthy [6]. The understandability of software knowledge is the ability to retrieve related and relevant information that matters to software development. For instance, an architect should be able to find related requirements and understand their impact on the quality attributes of the system. This information can be stored in different specification files, notes, and emails. They are distributed amongst different people and no one would possess all the knowledge. The key challenge in software knowledge retrieval is to find ways to relate the knowledge such that the knowledge an be used in a meaningful way to support software development.

III. INDEXING ARCHITECTURE KNOWLEDGE

To deal with the problems of creating and retrieving documented knowledge in software development, we propose to change the current practice of file-based software documentation. We propose to index software documents with a suitable lightweight ontology that is effective and easy-to-use. We have implemented a prototype system using a generic and adaptable ontology to support knowledge indexing and retrieval from software documentation. We draw an analogy between an ontology-based knowledge retrieval system and a relational database system, where the ontology is the index and software documents are the data contained in tables. The main difference between the two is that an ontology-based system supports reasoning and facilitates sharing of knowledge. In this section, we describe our approach to create such a knowledge indexing and retrieval system with some examples.

A. **A Lightweight Ontology**

An ontology defines a common vocabulary for those who need to organize and share information in a given domain. It is composed of machine-interpretable definitions of concepts in that domain and the relationships among them, which provide a natural foundation to organize the knowledge in linear software documents (e.g., requirements specifications, architecture design specifications sequentially documented in Office Word, etc.). These linear documents are created and modified with non-linear thoughts and activities in software development. E.g., a requirements engineer documents a new non-functional requirement whilst a software architect makes a related new design decision simultaneously. The concepts and relationships in an ontology can relate and organize this document information through reasoning (e.g., the *non-functional requirement* [a concept] is *affected by* [a relationship] a *design decision* [a concept] through *implementing by* [a relationship] *a component* [a concept]).

The other benefit of using an ontology is that it facilitates knowledge retrieval by automatic reasoning and querying with the support of formal semantics, which explicitly define and relate the use of software documentation related terms, such as requirements and architecture design concepts. An example is to query all the *components* that are used to *implement* a non-functuional *requirement*.

The potential drawback of using an ontology is that it requires effort for knowledge indexing. We address this issue in our prototype system in three ways: (a) use a lightweight ontology that can be tailored to suit retrieval requirements of architects; (b) provide a wiki-based click-and-select knowledge indexing tool; (c) semi-automate indexing for terminologies that the system already knows about.

We have created a lightweight ontology for indexing knowledge in requirements and architecture design documents, as shown in Figure 1. The concepts in this ontology (e.g., *Functional Requirement*) are commonly documented in requirements and architecture design specifications. We intend for such a lightweight ontology to be flexible and adaptable so that software organizations can adapt it for their own purposes. In order to support document identification and version control, we adapt and enhance Dublin Core (DC) definitions in our ontology. The initial concepts are described in [7].

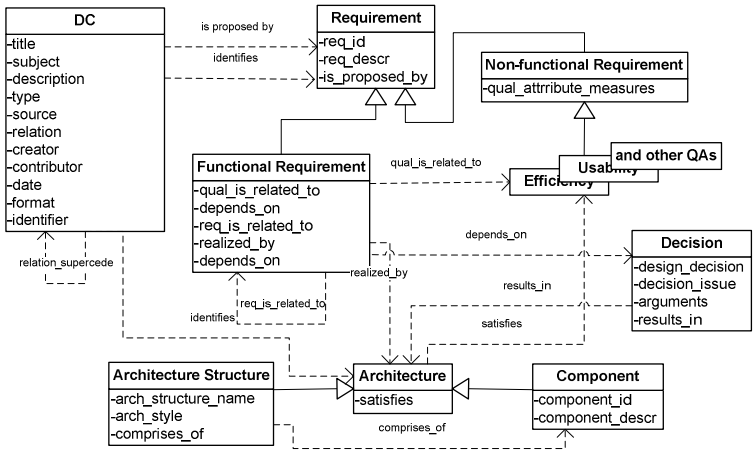

Figure 1. A lightweight ontology for indexing knowledge in requirements and architecture design documents

B. Using Ontology to Index Software Documents

Indexing software documentation using an ontology can be performed in two steps. Firstly, user may select some text (e.g., a phrase, sentence, or paragraph) from the document, and associate this text using select-and-click with an ontology concept. A knowledge instance of a concept is created and indexed in the knowledge base. Secondly, if a concept is related to another concept, a link can be established, again through select-and-click, between indexed knowledge instances. This is the process of creating knowledge *triples*.

We have implemented a prototype system based on OntoWiki (OW) [8] to support knowledge indexing and retrieval from software documents with wiki pages in OW. Users may highlight a requirement and indicate that this requirement is a *Functional Requirement* [a concept] and it belongs to a certain *Application Sub-system* [a concept]. This requirement is indexed by both concepts. If a user wants to indicate that a requirement is *realized by* [a relationship] an architecture design, a link between the two knowledge instances can be established through the OW user interface and a knowledge *triple* is created and stored in the knowledge base to indicate this relationship.

Users do not want to repeatedly index the same phrases that are already known in the knowledge base because it is time consuming. Thus the knowledge indexing activity should be semi-automated with the aid of some natural language processing techniques. In our system, any text that is found matching the pre-existing knowledge instances in the knowledge base is highlighted. Architects can use an user interface tool to confirm matched concepts. We also plan to provide automated facilities for version control, allowing semantic wiki-pages and the indexed contents to evolve.

C. Retrieving Knowledge

We present an example of knowledge retrieval from requirements and architecture documents with a concrete scenario in software development: Design Impact Evaluation.

Scenario Description: Requirements are frequently updated and added, and an architect needs to evaluate and identify the impact of the changing requirements on architecture design, so that requirements and architecture design remain consistent.

Knowledge Retrieval Example: A new requirement to support flexible pricing of a public transport system by time-of-day is considered. Pricing is currently defined in a pricing table within a database and it is implemented by an application component. When the architect queries "Pricing" requirements and the impacts of changing this requirement (e.g., using the ontology in Figure 1), the following knowledge is retrieved: (a) all existing requirements containing the word "Pricing" are retrieved; (b) all the designs that *realize* the "Pricing" requirements are retrieved; (c) all the non-functional requirements that are *satisfied-by* the design components are retrieved; (d) concerned stakeholders and version details are associated with each piece of information. The retrieved knowledge forms a network of trails. Following the knowledge trail, the architect discovers firstly that time-of-day is not defined in the existing database schema; secondly the current smart-card readers installed in buses to debit the cards have certain performance characteristics; thirdly, there are performance requirements of this device that must be satisfied. With the trail of related knowledge, an architect could evaluate the viability of the requirement change using important design relationships.

D. Preliminary Results

With the current version of our prototype system for indexing and retrieving knowledge from software documentation, we provide some traceability between requirements and architecture design using the lightweight ontology. The extent to which knowledge indexing and retrieval is effective largely depends on the ontology definition as well as the diligence of architects in indexing the knowledge. This in turn depends on the usability of a tool. We conjecture that the idea of dividing text from a file-based document into wiki pages which are shorter and readily updatable would encourage architects to document their updates and index their knowledge. The benefits of indexed knowledge during software development should pay for the indexing effort and be another incentive for architects to perform knowledge indexing.

The prototype system has been shown and discussed with a number of industry practitioners. Early and informal feedback from the architects indicates that this concept of creating semantic wiki pages and indexing knowledge with ontology could be useful in capturing knowledge that they sometimes do not document in specifications.

IV. Discussion and Conclusion

File-based software architecture documentation is ineffective in terms of knowledge capture and retrieval, especially in large and multi-site system development where it involves many distributed stakeholders and the system evolves over time. As such, we need to better understand what needs to be documented and how best to capture knowledge. We also need to understand how software people want to retrieve this knowledge. We envisage an environment in which multi-author, distributed-location, and continuous-updates are typical in software development. To support this environment, suitable methods and tools are badly needed to replace existing file-based documentation methods. The use of semantic wikis with suitable and flexible ontology appears to be useful and promising.

We have constructed a lightweight software engineering ontology and a prototype system to understand if this approach may provide a suitable solution. We have partially integrated and enhanced two tools to develop knowledge capture and search functionality: (a) OntoWiki to capture and index software documentation in wiki pages; (b) a semantic search and reasoning tool called ClioPatria [9] for advanced knowledge retrieval.

In evaluating how best to document and retrieve software architecture knowledge, we need to understand how software workers in an organization make use of software engineering knowledge and application domain-specific knowledge. Such an understanding will help us build a general ontology to enable knowledge indexing. Knowledge indexing using semantic wikis may potentially benefit many aspects of software development. It could improve software knowledge retention; facilitate communication between analysts, designers, and architects; and potentially reduce requirements and design inconsistency and omission. It could provide traceability of requirements and design; and provide better support for design reasoning and design verification. Finally, it could yield serendipitous insights, like in other semantic web applications. As this research is new, there are many opportunities and challenges ahead of us. Not all of the challenges are technology related, some of them are about how software people work and working together:

- **Knowledge capture** - in what form, at what level of detail, and at what cost of software documentation would encourage knowledge capture. What motivates people to contribute to documented and indexed knowledge?
- **Knowledge retrieval** - identify different ways in which software workers would retrieve knowledge, from the perspectives of: (a) software engineering process; (b) application domain; (c) developer's responsibility.
- **Knowledge tool implementation** - create technologies and usable tools to support effective creation and retrieval of software knowledge.

Acknowledgment

This research has been partially sponsored by the Dutch "Regeling Kenniswerkers", project KWR09164, Stephenson: Architecture knowledge sharing practices in software product lines for print systems, and the Natural Science Foundation of China under Grant No. 60950110352, STAND: Semantic-enabled collaboration Towards Analysis, Negotiation and Documentation on distributed requirements engineering.

References

1. Tang, A., Boer, T.d., Vlient, H.v.: Building Roadmaps: A Knowledge Sharing Perspective. Sixth Workshop SHAring and Reusing architectural Knowledge, Hawaii (2011)
2. Tang, A., Jin, Y., Han, J.: A rationale-based architecture model for design traceability and reasoning. Journal of Systems and Software **80** (2007) 918-934
3. Egyed, A., Grunbacher, P.: Supporting software understanding with automated requirements traceability. International Journal of Software Engineering and Knowledge Engineering **15** (2005) 783-810
4. Farenhorst, R., Lago, P., Vliet, H.v.: EAGLE: Effective Tool Support for Sharing Architectural Knowledge International Journal of Cooperative Information Systems **16** (2007) 413-437
5. Ali-Babar, M., Dingsøyr, T., Lago, P., Van Vliet, H. (eds.): Software Architecture Knowledge Management: Theory and Practice. Springer-Verlag Berlin Heidelberg (2009)
6. Jansen, A., Avgeriou, P., Ven, J.S.v.d.: Enriching software architecture documentation. Journal of Systems and Software **82** (2009) 1232-1248
7. Tang, A., Liang, P., Clerc, V., Vliet, H.v.: Supporting Co-evolving Architectural Requirements and Design through Traceability and Reasoning. In: Avgeriou, P., Lago, P., Grundy, J., Mistrik, I. (eds.): Relating Software Requiremens and Software Architecture (2011)
8. Agile Knowledge Engineering and Semantic Web (AKSW): OntoWiki:Semantic Collaboration for Enterprise Knowledge Management, E-Learning and E-Tourism. Vol. 2011 (2011)
9. Department of Computer Science: The ClioPatria semantic search web-server. Department of Computer Science, VU University Amsterdam (2010)