Towards Consistency Checking between Software Architecture and Informal Documentation

Jan Keim, Anne Koziolek
Karlsruhe Institute of Technology
Karlsruhe, Germany
{jan.keim, anne.koziolek}@kit.edu

***Abstract*—In the development process, documenting the software architecture is important to capture all reasoning and design decisions. Without a good and complete documentation, there is a lot of tacit knowledge that easily can get lost resulting in threats for success and increased costs. However, software architecture documentation is often missing or outdated. One reason for it is the tedious and costly process of creating and updating documentation in comparison to (perceived) low benefits. In this paper, we first present our long-term vision, where all information from any sources are persisted to avoid losing crucial information about a system. A base problem in this vision is keeping information from different sources consistent, with a major challenge of keeping consistency between models and informal documentation. We plan to address checking the consistency between models and textual natural language artefacts using natural language understanding. For this, we plan to break down the task into smaller subtasks that should provide a better understanding of the documents and their semantics. The extracted information should then be used to create traceability links and to check whether statements within the textual documentation are consistent with the software architecture models.**

***Index Terms*—Natural language processing, Software architecture, Software engineering, Consistency, Natural language understanding, Software architecture documentation**

I. Introduction

Software architecture plays a major role in the development, maintenance, and evolution of software systems. Every well-engineered software system has a good software architecture [1], so choosing a suitable architecture for software systems is important for the success of the system. Modelling the software architecture helps architects in the decision process to find a suitable architecture. Model facilitate communication and can enable simulation and prediction of quality attributes of a software system like performance [2]. This can be useful when comparing different design alternatives or for identifying potential bottlenecks early. Modern practical software development principles like agile and iterative development also state the importance of models [3], [4]. There, a main purpose of modelling is communication, but modelling is also seen to be key for iteration planning. For agile modelling, fundamental practices include the creation of several models in small increments [4].

Software Architecture Documentation (SAD) is a way to preserve knowledge about the software architecture and the underlying design decisions that led to the software architecture. Architects spend a significant amount of their time on expanding knowledge to come to good design decisions [5]. Documenting these decisions is important to avoid losing knowledge and to prevent the deterioration of these systems [6]. For example, in the context of open source software (OSS), research has shown that documentation indeed has a positive impact on the adoption of OSS and the economics of software development when using OSS [7]. Although software architecture documentation brings many benefits, there are some issues that we want to outline in the following.

a) Missing or outdated documentation: One major problem with SAD is that documentation is often missing or not updated regularly, causing it to become outdated. Without explicit up-to-date SAD, knowledge about a software system might perish. Additionally, incomplete documentation and missing knowledge in long-living systems result in the deterioration of these systems and in increased costs for maintenance and evolution [6]. Yet, Ding et al. showed that only 108 out of 2000 open source projects had some kind of documentation [8].

b) Consistency between artefacts: To avoid having any contradictory and conflicting information in different artefacts, it is important to have consistency between artefacts. Some approaches are already tackling consistency between models, e.g. [9], [10]. Besides models, there are more informal artefacts, e.g. software architecture documentation that is written in natural language. Using natural language texts for documentation is a common choice [8]. Although the freedom of expression of natural language comes with drawbacks like potential ambiguities, a major benefit of natural language is its accessibility and ease of use. However, there is a lack of research regarding consistency between models and such informal software architecture documentation.

A reason for these issues is that the tedious and costly process to create documentation and keep it up-to-date. Therefore, Ambler stated in [4] that the benefit must be greater than the cost of creating and maintaining it. Moreover, the actual benefits of documentation is often not seen by the creators, because the creators often do not rely on the documentation themselves. This is why it is important to research ways to reduce costs and improve benefits.

To tackle the stated problems, we aim to use natural language processing (NLP) and natural language understanding (NLU) techniques to analyse the natural language texts and compare the contained information with existing models and code. The core idea is to make NLP and NLU feasible in this context by

including knowledge in the form of ontologies about software architecture in general, about the domain of the software system at hand, and about the current software system itself to help gaining a thorough understanding. The ontologies about the current software system should be derived from existing models and code. A similar effort based on knowledge coming from ontologies constructed from the API of a system has been made with respect to programming in natural language with promising results [11]. Considering that modern software development is characterised by agile methods and iterative and incremental software development, we are confident to assume that models and code already exist for the system at hand and can be used for our approach.

The rest of the paper is structured as follows. In Section II we present our long-term vision for the software architecture development and documentation process. We outline in Section III our next goal about checking consistency, namely consistency between models and software architecture documentation written in natural language. An overview of related research is given in Section IV and we conclude this paper in Section V.

II. Long-term Vision

Meetings and discussions about the software architecture of a system are a central aspect in the development and evolution of software. Important information like design decisions and their reasoning are voiced in these discussions and in sketches on whiteboards. Yet, we do not often see all the information properly represented and persisted in the documentation of software architecture. Usually, only the results are fed into the implementation in the form of updates to the previous version; the ideas behind the changes are only implicitly present. All information not explicitly contained in these updates is tacit knowledge and might get lost.

In our long-term vision, we want to explore how to process and capture information gained through discussions in meetings, including whiteboard discussions. In a first step, we are interested to explore whether information can be extracted from spontaneous speech. For example, architects may utter statements about the architecture, such as assumed dependencies or interactions between components. In addition to make use of information in natural language, we plan to explore the inclusion of information contained in informal sketches in a second step. We also want to explore whether the combined data can be processed to automatically update models, so the collected information can be fed both into the software architecture and into its documentation.

If automatic updating of models will turn out to be feasible, an ad-hoc processing of such discussions could be used to provide quick feedback as an aid for discussions. On one hand, this feedback could contain information about possibilities and impossibilities of certain solution options regarding the constraints of the software system. On the other hand, the information could be processed to update models which can be used to predict certain properties of the system, e.g. quality properties like performance. With the additional information gained through these predictions, different design alternatives might be compared better to chose more fitting solutions.

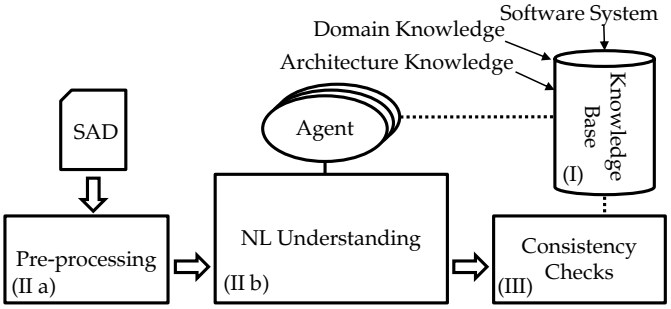

Fig. 1. Overview of planned approach, based on [12]

There are two major goals, we want to accomplish. Firstly, we want to tackle the problem of missing or insufficiently documented software architecture. We want to avoid throwing away any information and persist possibly all available information about a system to reduce the amount of tacit knowledge and to reduce the chance of losing important information. An important part of this involves providing an easy and fast access to the appropriate information that a user is interested into. Secondly, we want to reduce the overhead to create documentation and models to support the architect by automating (parts of) the needed processes. This way, the architects can focus on the creation of the actual architecture instead of spending lots of time on updating models and documentation. Additionally, this might encourage to actually create and maintain documentation.

III. Checking Consistency

One problem for SAD we mentioned earlier is the consistency between different artefacts. Our idea is to tackle automated consistency checking between models and informal documentation in the form of natural language texts. We plan to use various NLP techniques and incorporate knowledge bases.

An overview of our plan to realise our idea is depicted in Figure 1 and can be divided into three major parts: (I) creation of a knowledge base, (II a) pre-processing of the input SAD texts followed by (II b) the main processing to generate a thorough understanding of the text, and (III) checking the consistency between the input documents and the system's architecture in the post-processing step.

The creation of the knowledge base for part (I) takes information from different levels into account. We currently see three different levels of knowledge. Firstly, there is general architecture knowledge containing concepts such as components, interfaces, and associations. Additionally, knowledge about specific software architecture styles like microservice architecture falls into this category. Secondly, knowledge about the domain of the software system, i.e. knowledge about the business domain, should be included. This can help in different processing steps, e.g. word sense disambiguation can be improved by giving domain-specific senses more weight. Lastly, knowledge about the software system including the

current software architecture and its models, preferably even existing code, needs to be added into the knowledge base.

In step (II) we aim to gain understanding about the texts. The framework we plan to use for this step is be based on the agent-based framework ProNat [12], [13]. We plan to begin with a pre-processing step (II a) that prepares the input for the NLU and performs basic NLP tasks, e.g. parsing, chunking, and sentence splitting. After this, we can do the main processing step (II b) that tackles NLU. There, we intend to break down the complex task into smaller subtasks that will be solved using different agents. In our experience, this enables us to use more general approaches and to use existing state-of-the-art techniques for each subtasks. Besides splitting into subtasks, the usage of agents also allows their independent execution. Still, agents can use information generated by other agents and can run multiple times to improve their results based on other agents' results. For example word sense disambiguation is needed to identify topics within the document, but the identified topics might also improve results of the disambiguation of single words that got disambiguated wrongly. Example tasks for agents also include coreference resolution and named entity recognition. Further, agents can operate either domain-agnostic or domain-aware using the knowledge base.

The post-processing step of part (III) uses the gathered information to check consistency. With the check we want to determine if the documentation contradicts the models in any way. For this, the previously extracted information is used to query the knowledge base about the software system. The results of these queries are then evaluated for any discrepancies. For example, the documentation might state that a connection between two components exists using a specified interface. If there is no connection or the stated interface cannot be matched with the actually used interface, an inconsistency is found and the user, i.e. the software architect, is notified.

For us, a major goal of our approach for checking consistency is the reduction of time-consuming manual work in this area. This may reduce the human workload and the amount of undesirable work and may support the software architect to recognize and update inconsistencies. Besides reduction of manual work, we also see other benefits. Firstly, consistency between models and software architecture documentation is tackled. This is especially helpful in iterative development and evolution scenarios, where software including its documentation and architecture needs to be updated regularly. Also, this may help reducing the overhead needed for software architects to create either documentation or models. Especially after updating the software architecture, it is important to also update the documentation and check for (in-) consistency. Additional benefits may arise, when this approach is combined with tools that can perform transformations between code and software architecture models, e.g. SoMoX [14]. Incorporating a code-to-architecture-transformation could enable architecture conformance checks between SAD and the implementation via the generated software architecture models.

Moreover, our plan for checking consistency involves a generation of traceability links between documentation and models. This tackles another problem of SAD as creating and maintaining traceability links is still a challenge [15]. The created traceability links may increase knowledge and the understanding about the system.

IV. Related Work

There are various research directions that are related to the underlying problem for our approach.

A close research direction deals with conformance checking between software architecture and code. For example Schröder and Riebisch [16] developed an approach to check conformity using reasoning on a knowledge base that is created out of architecture rules in controlled natural language (CNL) combined with code. In contrast, our approach aims for consistency between software architecture and natural language SAD. Natural language is easier to use than CNL and a common choice for SAD [8].

Traceability is another closely related research topic that usually covers traceability between requirement and source code. Concepts and ideas to create traceability links between requirements and source code might be helpful and mutual problems do exist. Zheng et al. developed an approach that creates traceability links between features of software product lines and source code through product line architecture using an extended architecture description language to describe product features [15]. Other research about traceability focuses on the analyses of structure and dependencies of source code [17]–[19]. Many approaches like these use information retrieval methods to create traceability links. Compared to our idea, they do not use NLU to gain a better understanding of the underlying information. Although there are approaches that consider semantics [20], these approaches do not use the full potential of NLU techniques.

Another closely related research direction deals with consistency between models, usually by using transformations between models. One approach within this direction is the VITRUVIUS approach [21]. Burger et al. aim to generate a virtual single underlying meta-model (VSUMM) that keeps consistency between models and allows accessing the models via views. Changes to a model of a concrete virtual single underlying model (VSUM) are propagated with change-driven incremental transformations to preserve the consistency [9]. This and other approaches in this research direction have the same goal of checking and preserving consistency. While these approaches tackle model consistency, we aim to tackle consistency between natural language texts and models instead.

A further related research direction for our idea is about SAD and the documentation of design decisions. Kruchten described a classification of design decisions and provided an ontology to capture said design decisions [22]. He states that design decisions are both explicitly and implicitly visible in the resulting software architecture, where for example non-existence is only implicitly visible. To understand the full reasoning behind a software architecture, the implicit knowledge needs to be made explicit. [23] tackles this by introducing architectural knowledge repositories. Alexeeva et

al. [24] give an overview of literature tackling design decision documentation including publications aimed at consistency and compliance of decisions and architecture. Yet, we mainly see the focus on traceability and barely on consistency between architecture and design decisions.

Lastly, the research area of natural language processing, especially for software engineering, is important in our context. The PARSE project [25] set the goal to generate script-like programs out of spoken explanation [13]. For this, the agent-based framework ProNat [12] was developed. Despite the differences, there are many shared problems and similar difficulties, thus we plan to use parts of this research. The problem of mapping natural language texts to models in general is also tackled by approaches that try to create (UML) models out of requirements and similar. Some approaches use structural analyses [26] or semantic roles [27] to resolve this problem. Although the creation of models is different from comparing the texts with existing models, underlying concepts of the structural analyses or considering semantic roles can be useful.

Although all the mentioned research directions are in one way or another related to our goals, we do not see consistency between textual SAD and software architecture models properly tackled yet.

V. Conclusion

In this paper, we presented our idea to improve software architecture documentation. We explained that an important benefit of software architecture combined with good documentation is the reduction of tacit knowledge. To tackle this problem, we presented our long-term vision, where we want to explore how to capture, understand, and persist what software architects explain and discuss on whiteboards. All information in this process should be be persisted to document design decisions and reasoning. Additionally, models might be updated automatically to predict quality attributes like performance.

We showed our next goal that consists of checking consistency between software architecture models and informal software architecture documentation in form of natural language text, which we did not see properly covered in existing research. We outlined our planned approach that uses an agent-based framework to break down the task into subtask and incorporates knowledge from different sources. The expected benefits include consistent documentation and increased traceability but also a reduction in the human workload.

References

[1] N. Medvidovic and R. N. Taylor, "Software architecture: Foundations, theory, and practice," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 471–472.

[2] R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann, *Modeling and simulating software architectures: The Palladio approach*. MIT Press, 2016.

[3] S. W. Ambler and M. Lines, *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise*. IBM Press, 2012.

[4] S. W. Ambler, "Agile modeling," http://agilemodeling.com/.

[5] R. Farenhorst and H. van Vliet, "Understanding how to support architects in sharing knowledge," in *ICSE Workshop on Sharing and Reusing Architectural Knowledge*, 2009, pp. 17–24.

[6] D. L. Parnas, "Software aging," in *Proceedings of the 16th International Conference on Software Engineering*, ser. ICSE '94. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 279–287.

[7] S. A. Ajila and D. Wu, "Empirical study of the effects of open source adoption on software development economics," *Journal of Systems and Software*, vol. 80, no. 9, pp. 1517–1529, 2007.

[8] W. Ding, P. Liang, A. Tang, H. v. Vliet, and M. Shahin, "How do open source communities document software architecture: An exploratory survey," in *19th International Conference on Engineering of Complex Computer Systems*, 2014, pp. 136–145.

[9] M. E. Kramer, M. Langhammer, D. Messinger, S. Seifermann, and E. Burger, "Change-driven consistency for component code, architectural models, and contracts," in *18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*, 2015, pp. 21–26.

[10] H. Klare, "Multi-model Consistency Preservation," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS 2018*, October 2018, pp. 156–161.

[11] M. Landhäußer, S. Weigelt, and W. F. Tichy, "NLCI: a natural language command interpreter," *Automated Software Engineering*, vol. 24, no. 4, pp. 839–861, December 2017.

[12] S. Weigelt and W. F. Tichy, "Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language," in *IEEE International Conference on Software Architecture*, 2015, pp. 819–820.

[13] S. Weigelt, T. Hey, and W. F. Tichy, "Context model acquisition from spoken utterances," *International Journal of Software Engineering and Knowledge Engineering*, vol. 27, no. 09n10, pp. 1439–1453, 2017.

[14] O. Travkin, M. Von Detten, and S. Becker, "Towards the combination of clustering-based and pattern-based reverse engineering approaches." in *Software Engineering (Workshops)*, 2011, pp. 23–28.

[15] Y. Zheng, C. Cu, and H. U. Asuncion, "Mapping features to source code through product line architecture: Traceability and conformance," in *IEEE International Conference on Software Architecture*, 2017, pp. 225–234.

[16] S. Schröder and M. Riebisch, "An ontology-based approach for documenting and validating architecture rules," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ser. ECSA '18. ACM, 2018, pp. 52:1–52:7.

[17] H. Kuang, P. Mäder, H. Hu, A. Ghabi, L. Huang, J. Lü, and A. Egyed, "Can method data dependencies support the assessment of traceability between requirements and source code?" *Journal of Software: Evolution and Process*, vol. 27, no. 11, pp. 838–866, 2015.

[18] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "When and how using structural information to improve IR-based traceability recovery," in *17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 199–208.

[19] J. I. Maletic and A. Marcus, "Supporting program comprehension using semantic and structural information," in *Proceedings of the 23rd International Conference on Software Engineering*, ser. ICSE '01. IEEE Computer Society, 2001, pp. 103–112.

[20] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," *Requirements Engineering*, vol. 20, no. 3, pp. 281–300, September 2015.

[21] E. J. Burger, "Flexible views for view-based model-driven development," in *Proceedings of the 18th International Doctoral Symposium on Components and Architecture*, ser. WCOP '13. ACM, pp. 25–30.

[22] P. Kruchten, "An ontology of architectural design decisions in software intensive systems," in *2nd Groningen workshop on software variability*. Citeseer, 2004, pp. 54–61.

[23] P. Kruchten, P. Lago, and H. v. Vliet, "Building up and reasoning about architectural knowledge," in *Quality of Software Architectures*, ser. Lecture Notes in Computer Science. Springer, 2006, pp. 43–58.

[24] Z. Alexeeva, D. Perez-Palacin, and R. Mirandola, "Design decision documentation: A literature overview," in *Software Architecture*, ser. Lecture Notes in Computer Science. Springer, Cham, 2016, pp. 84–101.

[25] S. Weigelt, "PARSE – Programming ARchitecture for Spoken Explanations," https://parse.ipd.kit.edu/.

[26] O. Keszocze, M. Soeken, E. Kuksa, and R. Drechsler, "Lips: An IDE for model driven engineering based on natural language processing," in *1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, 2013, pp. 31–38.

[27] T. Gelhausen and W. F. Tichy, "Thematic role based generation of UML models from real world requirements," in *International Conference on Semantic Computing (ICSC)*, 2007, pp. 282–289.