Towards Consistency Analysis between Formal and Informal Software Architecture Artefacts

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

***Abstract*—Documenting the architecture of a software system is important, especially to capture reasoning and design decisions. A lot of tacit knowledge can easily get lost when the documentation is incomplete, resulting in threats for the software system's success and increased costs. However, software architecture documentation is often missing or outdated. One explanation for this phenomenon is the tedious and costly process of creating documentation in comparison to (perceived) low benefits. In this paper, we first present our long-term vision, where we plan to persist information from any sources, e.g. from whiteboard discussions, to avoid losing crucial information about a system. A core problem in this vision is the possible inconsistency of information from different sources. A major challenge of ensuring consistency is the consistency between formal artefacts, i.e. models, and informal documentation. We plan to address consistency analyses between models and textual natural language artefacts using natural language understanding and plan to include knowledge bases to improve these analyses. After extracting information out of the natural language documents, we plan to create traceability links and check whether statements within the textual documentation are consistent with the software architecture models. In this paper, we also outline our requirements for evaluating our approach with the help of a community-wide infrastructure and how our approach can be used to maintain community-wide case studies.**

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

I. Introduction

The architecture of a software system plays a key role for the success of the software system. According to [1], every well-engineered software system has a good architecture. Thus, choosing a suitable architecture is important for the development, maintenance, and evolution of the system system. To find a suitable architecture, modelling the system can help. Models improve communication capabilities and can enable simulation and prediction of software quality attributes like performance [2]. Modern software development principles emphasize the importance of modelling [3], [4]. The importance of modelling is especially emphasized for communication reasons, but is also seen as key element for iteration planning. For agile modelling, a practice-based methodology for effective modelling and documentation of software-based systems, the creation of models in small elements is fundamental [4].

Knowledge about the software architecture of a system is important and preserving said knowledge is a key element for Software Architecture Documentation (SAD). Architects spend a significant part of the software architecture design process on expanding their knowledge to come to good design decisions [5]. Losing knowledge about the system usually causes its deterioration [6] and a method to avoid deterioration is documenting said knowledge, including design decisions. When design decisions and reasoning that lead to a certain architecture are documented, the effort of gaining specific knowledge about the software system also does not have to be made again. Additionally, in [7] the authors showed that there is a positive impact of documentation on the adoption of open source software (OSS). Although software architecture documentation brings a lot of benefits, we stated in [8] some issues regarding current software documentation that are outlined in the following.

a) Missing or outdated documentation: A common problem with SAD is the absence of documentation. Besides the frequent absence of documentation, irregular updates also cause the documentation to become outdated. Because of missing up-to-date SAD, knowledge about a software system tends to decline. As mentioned earlier, incomplete documentation and missing knowledge results in the deterioration of software systems and increased costs for maintenance and evolution [6], especially for long-living systems. However, Ding et al. showed in [9] that only 108 out of 2000 open source projects had some kind of documentation.

b) Consistency between artefacts: Consistency between artefacts is important to avoid having any contradictory or conflicting information in different artefacts. Some approaches are already tackling consistency between models, e.g. [10], [11]. When inconsistencies can be, preferably automatically, found and resolved, architects and developers can operate on different artefacts but still contribute to the same underlying system. Thus, they can work with artefacts they are familiar with and that are most suitable for their intentions. However, consistency between models and informal software architecture documentation is barely researched at all. Such informal artefacts include documentation written in natural language. Natural language texts are a common choice for documentation [9], because natural language is accessible and easy to use. However, its freedom of expression also has drawbacks like potential ambiguities and less suitability for automated processing.

A main reason for these problems is the tedious and costly process of creating documentation and keeping it up-to-date. The (perceived) benefit of documentation must be greater than

the cost of creating and maintaining it [4]. Moreover, software architects who created the documentation usually do not rely on the documentation themselves, reducing the perceived benefit of creating documentation. It is important to research ways that benefits both parts, reducing costs and improving benefits.

To approach the stated problems, we aim to use natural language processing (NLP) and natural language understanding (NLU) techniques. We plan to analyse natural language documents to extract the contained information about software architecture. We want to use the extracted information to compare it with existing models and code of the software system. To make NLP and NLU feasible in this context, we plan to incorporate knowledge bases using ontologies with different types of knowledge: Knowledge about software architecture in general, domain-specific knowledge about the software system, and project- and system-specific knowledge about the current software system itself, i.e. models and code. We expect this to provide our approach the possibility to gain a more thorough understanding. A similar effort based on knowledge coming from ontologies has been made with respect to programming in natural language with promising results [12], where ontologies were constructed from the API of a system. Considering that modern development is characterised by agile methods and is usually performed in an iterative and incremental manner, we are confident to assume that models and code already exist, thus can be used in our approach.

Case studies play an important role for our research, both for analysing typical structures of software architecture documentation as well as for evaluation purposes. This is why we are interested in a community-wide infrastructure that can provide a centralized access to case studies. Besides that, we also believe that our planned approach can benefit the community.

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. In Section IV we discuss our needs to be able to conduct proper evaluation of our approach using community-wide case studies and how our approach can benefit a community-wide infrastructure. An overview of related research is given in Section V and we conclude this paper in Section VI.

II. Long-term Vision

Discussions about the architecture of a system under development, e.g. in meetings, are a key aspect of development as well as evolution of software. In these discussions, important information about the software like design decisions and their reasoning are discussed and sketches of the software architecture are drawn on whiteboards and similar. Although these discussions are important and often contain a lot of essential information for the success of a software systems, most information within these discussion are not documented. Most of the time, only the results are persisted directly into the software; reasoning and alternatives are not recorded in the documentation.

Our long-term vision, as stated in [8], is to explore how we can improve this situation. We plan to explore ways of capturing the information gained through discussions in meetings, including whiteboard discussions. Besides these face-to-face discussions, we also think about the inclusion of other forms of communication like mailing lists or issue trackers. For this, we want to explore in a first step whether all necessary information can be extracted from spontaneous speech and similar communication. For example, architects may express relations like interactions or assumed dependencies between components. In a second step, we want to explore the inclusion of informal sketches into the processing to also include information contained in these. Additionally, we want to explore whether all collected information can be automatically processed to be fed both into the documentation and the software architecture, i.e. into software architecture models.

If automatic updates of models are feasible, ad-hoc processing of discussions can be used to provide quick feedback. This feedback can aid discussions by providing information about possibilities and impossibilities of certain options and solutions regarding constraints of the software system under development. Additionally, these models can be used to predict further system properties, e.g. performance. This is especially useful to directly evaluate different design alternatives to find a suitable solution.

With our approach, we want to accomplish the two major goals mentioned in [8]. Firstly, the problem of missing and insufficiently documented software architecture should be approached. Currently, information about alternative design decisions or reasoning is often not persisted but instead thrown away. We want to reduce the amount of tacit knowledge and we want to reduce the chance of losing any important information by simply persisting more information. This also creates the need for an easy and fast access to the appropriate information that a user is interested into to avoid actually reducing the usefulness and handiness because users are overloaded with information. Secondly, we want to reduce the overhead to create software architecture documentation and models to support the architect. This can be realized by automating needed processes or parts of these processes. This way, architects won't need to spend lots of time with updating models and documentation but can focus on the creation of the actual architecture and on exploring alternative solutions. Additionally, this might encourage to actually create and maintain documentation, thus improve the overall development and maintenance process.

III. Consistency Analysis

One problem for software development is consistency between artefacts. If one artefact changes, the changes also need to be transferred to other artefacts to ensure consistency between artefacts. Especially in model-driven development, there are different models with different views on the same underlying information. There is already a research direction that deals with the problem of consistency between models. In

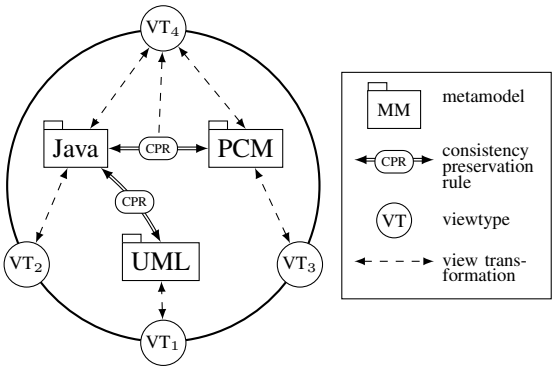

Figure 1. Example of a VSUMM in VITRUVIUS [13]

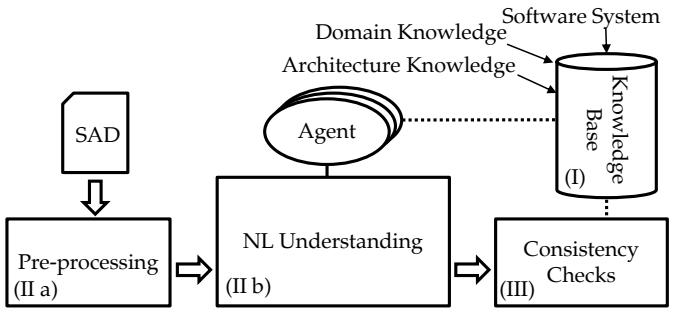

Figure 2. Overview of planned approach, based on [16]

this direction, common approaches are typically using transformations between models. One approach within this direction is the VITRUVIUS approach [14]. A goal of the VITRUVIUS approach is the generation of a virtual single underlying metamodel (VSUMM) that ensures consistency between models and allows accessing the models via views. An example for a VSUMM can be seen in Figure 1, where consistency preservation rules exist between Java and UML and between Java and the Palladio Component Model (PCM), a software component model [15]. Changes to a model of a concrete virtual single underlying model (VSUM) are propagated with change-driven incremental transformations to preserve consistency [10]. Within this direction falls the problem we want to tackle: preserving consistency between models and informal artefacts. Software architecture documentation can be seen as a special view on a software system. When documentation is updated, models need to be updated as well to reflect the adaptations to the documentation. Similarly, when the software system updates, e.g. because of an update to a model, there is the possibility of inconsistent documentation. We see the consistency problem as one of the reasons, why documentation often is not done or why documentation deteriorates. In the following, you can find two examples for statements in documentations that might cause inconsistencies:

- "Component A is connected to component B via the Interface I."
  The architecture of the system initially matched this statement. However, the connection between component A and B was removed while refactoring the system. This causes an inconsistency if the documentation is not updated accordingly.
- "Component A and component B have to be deployed on different machines."
  This constraint brings another possible kind of problem regarding inconsistency. When it is violated in a later development cycle, the inconsistency needs to be resolved and either the violation needs to be repaired or the original constraint needs to be updated.

This is why documentation, like other models, needs to be reviewed and updated when changes happen. However, there is the tendency to write less documentation due to consistency problems to avoid the overhead of updating the documentation conscientiously.

A. Approach

As described in [8], our idea is to tackle automated consistency analysis between models and informal documentation. In a first approach, we want to limit informal documentation to natural language texts. We plan to combine NLP techniques with knowledge bases about software architecture, the domain of the software, and the software itself. An overview of our planned approach is depicted in Figure 2. The approach can be divided into three major parts: (I) creation of a knowledge base or update of an existing one, (II) processing of the natural language input that is split into (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) analysing consistency by performing consistency checks between the analysed natural language input and the system's architecture. This process should run every time an artefact, i.e. documentation, architecture, or implementation, changes to ensure consistency in each increment of a system.

The knowledge base in part (I) takes information from three different levels into account. On the first level, there is general knowledge about software architecture containing concepts such as components, interfaces, and associations. This general knowledge is extended with knowledge about specific software architecture styles like microservice architectures. The second level contains knowledge about the domain of the software system, i.e. knowledge about the business domain. This knowledge can be used to support different processing steps. For example word sense disambiguation can be improved by giving domain-specific senses more weight when examining different meanings. The third level includes knowledge about the current software system, especially about the current software architecture and its models and preferably even existing code. Overall, parts of this knowledge base can be compared to ideas by Yuan [17] to create an ontology-based software architecture representation. The general parts of the knowledge base, namely the general knowledge about software architecture from level one and partly the domain-knowledge from level two, can be shared with other projects. This also means that a comprehensive ontology is not only useful for

our approach but might be useful for the software architecture and software development community in general.

The purpose of part (II) is to gain understanding about the input texts, i.e. the natural language software architecture documentation. In this part, we plan to use a framework that is based on the agent-based framework ProNat [16], [18]. The processing begins with a pre-processing task (II a) that performs basic NLP tasks like parsing, chunking, and sentence splitting. Moreover, the input is transformed into a graph representation to prepare the text for the main processing (II b). The main processing tackles the NLU. Here, we intend to break down the complex task into smaller subtasks. Each subtask will be solved using a different agent. In our experience, this enables the usage of more general approaches as well as the usage of existing state-of-the-art techniques for each subtask. Additionally, the usage of agents allows their independent execution. Still, agents are intended to use information generated by other agents to either enable their processing or to improve their results. Agents can also run multiple times to improve their results, e.g. based on new information provided by other agents. For example, word sense disambiguation is needed to identify topics within the document, but the identified topics might also improve the disambiguation results, e.g. for wrongly disambiguated words. Example tasks for agents also include coreference resolution and named entity recognition. By providing access to the knowledge base, agents can also become domain-aware.

Part (III) is used to process the previously gathered information to check consistency. Using the gathered information as well as reasoners and inference, the knowledge base is evaluated for any discrepancies. For example, the documentation might state that a specified interface is used by a component to interact with another component. If there is no connection or the stated interface cannot be matched with actually existing interfaces, an inconsistency is found and the user, i.e. the software architect, is notified. In our current plan, we do not want to automatically resolve found inconsistencies yet.

B. Planned procedure

We briefly want to outline our intended procedure and the next steps to realize our approach. A central aspect is the iterative approach. Development on the three parts depicted in Figure 2 will be done fairly simultaneously while each part will be improved iteratively. We want to get results early that can be used to decide on further necessary actions. Additionally, dead ends or less promising strategies can be detected early on and adjustments to our strategies can be made.

For the creation of the knowledge base, we plan to split the knowledge base into different, mostly independent ontologies. We also want to start with a simple and rather small knowledge base to enable early prototypes of our approach. The knowledge base will then be extended continuously. The general knowledge will be based on ontology-versions of existing metamodels like UML or PCM as well as metamodels for code, e.g. from JaMoPP[1] or MoDisco[2]. We plan to derive these ontologies by transforming existing Ecore[3]-based metamodels into ontologies. The system knowledge itself will then be represented by instantiating the metamodels. We also plan to extend the base ontologies with further needed attributes and properties, e.g. to improve natural language processing and understanding, as well as further relations, e.g. to express relations between PCM and UML.

An important step to check consistency is the creation of (traceability) links between text elements and elements within the knowledge base. To have an early prototype, we plan to start with simple links that can directly be derived from same or similar naming. Successively, we plan to improve the discovery of these links, e.g. by including synonyms as well as further domain knowledge. As mentioned before, to process the text we want to use the PARSE framework, where we can reuse some already existing agents, e.g. an agent for named entity recognition. New features like discovery of synonyms can then be added by implementing new agents.

As soon as links are present, we plan to start the actual consistency analyses. For these analyses, we need to research the best ways. One method we want to explore could be based on queries to the knowledge base. The queries could be generated with the extracted information about relations or kind of design decision in the input texts. A second method could be based on inference and reasoners on the ontologies. We also want to research, if a combination of different methods is more suitable.

C. Benefits

In [8], we described the reduction of time-consuming manual work as a major goal of our approach. The human workload as well as the amount of undesirable work may be reduced and our approach might support the software architect to recognise and update inconsistencies. We also see further benefits besides the reduction of manual work. Firstly, consistency between models and software architecture documentation is tackled. In iterative development and evolution of software, where software including its architecture and documentation needs to be updated regularly, consistency checks are particularly helpful. Our approach may also help reducing the overhead needed for software architects to create and update either documentation or models. When documentation or models are updated, the respective other artefacts also need to be updated and consistency between these should be ensured. Additional benefits may arise, when this approach is combined with tools and frameworks that can perform transformations between code and software architecture models, e.g. within the VITRUVIUS approach or with SoMoX [19] and similar tools. Incorporating such a code-to-architecture-transformation could also enable architecture conformance checks between SAD in natural language and the implementation via the generated software architecture models.

Moreover, our current plan involves a generation of traceability links between documentation and models. This tackles

[1]https://github.com/DevBoost/JaMoPP

[2]https://www.eclipse.org/MoDisco/

[3]https://wiki.eclipse.org/Ecore

another problem of SAD as creating and maintaining traceability links is still a challenge [20]. The created traceability links may also increase knowledge and understanding about the system.

IV. Connection to a community-wide infrastructure

In this section, we want to outline our requirements for evaluating our approach with the help of a community-wide infrastructure and how our approach can be used to maintain community-wide case studies.

In order to validate the proposed approach, we plan to use various case studies. Several different case studies already exist and each one offers different properties and advantages for different application fields. Common case studies in the field of software architecture include RUBiS[4], SPECjEnterprise[5], and Sock Shop[6].

For our approach, one of the requirements to use case studies for evaluation is an existing documentation that describes textually the software architecture using natural language. Preferably, the documentation should also include the reasoning behind architectural decisions together with their context and consequences. It would also be preferable, if a case study was accompanied by an available GIT history and issue trackers, so that other forms of natural language communication could also be considered. While looking for such case studies, we realized that they are hard to find. Although documentation is one of the recommended practices for improving development and maintenance support [21], open source software often either does not have any or only shallow documentation [9]. Furthermore, [22] showed that incomplete or confusing documentation is the biggest problem with open source software. Two examples for suitable case studies that we found are TeaStore[7] and TEAMMATES[8].

For this reason, a central collection of maintained case studies would be advantageous. They should particularly contain models, implementation, and (natural language) documentation. Such a collection is not only advantageous for us, but is also beneficial to the whole software architecture research community. Among other things, this would make it easier to find and select suitable case studies. For us, an important motive to have such a central collection is the possibility to evaluate the generalizability and robustness of our approach. At the same time, sharing case studies enables a proper comparison of evaluation results of different approaches. Furthermore, having an increasing amount of case studies would also enable the creation of a natural language corpus dedicated to software architecture. A dedicated software architecture documentation corpus would make it possible to develop new machine learning approaches or the adapt existing ones that are coined for the processing of software architecture documents. Because different approaches might need various kinds of artefacts of a system, i.e. different kinds of models, the case studies probably need to be (jointly) enriched with these artefacts. Artefacts of the software system might include UML (component) models as well as Palladio component models and similar. To ensure consistency between the different artefacts, approaches like the previously mentioned VITRUVIUS approach can be used.

Our approach can also help to maintain this central collection of community-wide case studies. Researchers usually need to get to know and understand the architecture and structure of a case study to be able to decide if a case study is applicable and suitable. For this and other purposes, documentation is useful and important. Having consistent documentation for a given software system and its architecture is therefore key to a functioning case study infrastructure. Especially, when a case study evolves or needs to be maintained, changes to the software and the architecture might happen, resulting in possibly outdated documentation. Our approach can help to maintain the documentation and ensure consistency.

V. Related Work

The underlying problems for our approach, including closely related problems, can be found in various research directions.

Schröder and Riebisch [23] tackle a similar problem that is conformance checking between software architecture and source code. With their approach, the software architect can state architecture rules in a special controlled natural language. A knowledge base is then created by combining an ontology derived from these architecture rules with an ontology derived from source code. Architecture violations are then identified using reasoning on this knowledge base. One major difference to our planned approach is the targeted conformance. The approach by Schröder and Riebisch targets conformance between software architecture and code while we aim to check consistency between software architecture documentation and software architecture. Additionally, their approach processes controlled natural language, while we aim to support natural language as it is easier to use and a common choice for SAD [9].

The research direction covering traceability that usually spans requirements and source code is also a closely related research topic. Concepts and ideas to create these traceability links 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 [20]. Other research about traceability focuses on the analyses of structure and dependencies of source code [24]–[26]. These approaches, like many others, 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 [27], these approaches do not use the full potential of NLU techniques. They do apply certain information retrieval methods that are more semantical,

[4]http://rubis.ow2.org/

[5]https://spec.org/jEnterprise2018web/

[6]https://github.com/microservices-demo/microservices-demo

[7]https://github.com/DescartesResearch/TeaStore

[8]https://github.com/TEAMMATES/teammates

but they do not aim to gain considerable and proper natural language understanding.

Another related research direction covers SAD and the documentation of design decisions. Kruchten describes in [28] a classification of design decisions and provided an ontology to capture said design decisions. 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. The architectural knowledge repositories introduced in [29] try to tackles this problem. Alexeeva et al. [30] give an overview of literature addressing design decision documentation including publications aimed at consistency and compliance of decisions and architecture. Yet, consistency between architecture and design decisions is, in our opinion, barely tackled and focus often lies on traceability.

Lastly, the research area of natural language processing, especially for software engineering, is important in our context. PARSE [31] is a project with the goal to transform spoken explanation into script-like programs [18]. The agent-based framework ProNat [16] was developed to approach this goal. As there are many shared problems and similar difficulties, we plan to use parts of the research for PARSE. The problem of mapping natural language texts to models in general is also tackled by approaches that try to create models, i.e. UML models, out of requirements and similar. Approaches are using structural analyses [32] or semantic roles [33] 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.

All the mentioned research directions are in one way or another related to our goals or related to parts of our goals. However, we do not see consistency between SAD in natural language texts and software architecture models properly tackled yet.

VI. 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. In this process, as much information as possible should 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 whole 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. We discussed our requirements to be able to properly evaluate our ideas, i.e. the requirement to have a collection of case studies. In the discussion we briefly highlighted possible benefits of such a collection, both for us and for the community. Moreover, we outlined, how our approach might be beneficial for the maintenance of said collection.

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, J. Henss, and M. Kramer, "Introduction," in *Modeling and Simulating Software Architectures – The Palladio Approach*, R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann, Eds. Cambridge, MA: MIT Press, October 2016, ch. 1, pp. 3–15.

[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] J. Keim and A. Koziolek, "Towards consistency checking between software architecture and informal documentation," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2019, accepted, to appear. [Online]. Available: https://sdqweb.ipd.kit.edu/publications/pdfs/keim2019nemi.pdf

[9] 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.

[10] 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.

[11] 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.

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

[13] J. Meier, H. Klare, C. Tunjic, C. Atkinson, E. Burger, R. Reussner, and A. Winter, "Single underlying models for projectional, multi-view environments," in *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*. SCITEPRESS, 2019, to appear.

[14] 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.

[15] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, and M. Kuperberg, "The Palladio Component Model," KIT, Fakultät für Informatik, Karlsruhe, Tech. Rep., 2011. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000022503

[16] S. Weigelt and W. F. Tichy, "Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language," in *37th IEEE International Conference on Software Engineering (ICSE)*, vol. 2, 2015, pp. 819–820.

[17] E. Yuan, "Towards ontology-based software architecture representations," in *1st IEEE/ACM International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, ECASE@ICSE*. IEEE, 2017, pp. 21–27.

[18] 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.

[19] 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.

[20] 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*, April 2017, pp. 225–234.

[21] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information*. ACM, 2005, pp. 68–75.

[22] F. Zlotnick, "Github open source survey 2017," http://opensourcesurvey.org/2017/, Jun. 2017.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.