Establishing a Benchmark Dataset for Traceability Link Recovery between Software Architecture Documentation and Models

Dominik Fuchß[0000−0001−6410−6769], Sophie Corallo[0000−0002−1531−2977], Jan Keim[0000−0002−8899−7081], Janek Speit, and Anne Koziolek[0000−0002−1593−3394]

KASTEL – Institute of Information Security and Dependability
Karlsruhe Institute of Technology, Karlsruhe, Germany
{dominik.fuchss,sophie.corallo,jan.keim,koziolek}@kit.edu

Abstract. In research, evaluation plays a key role to assess the performance of an approach. When evaluating approaches, there is a wide range of possible types of studies that can be used, each with different properties. Benchmarks have the benefit that they establish clearly defined standards and baselines. However, when creating new benchmarks, researchers face various problems regarding the identification of potential data, its mining, as well as the creation of baselines. As a result, some research domains do not have any benchmarks at all. This is the case for traceability link recovery between software architecture documentation and software architecture models. In this paper, we create and describe an open-source benchmark dataset for this research domain. With this benchmark, we define a baseline with a simple approach based on information retrieval techniques. This way, we provide other researchers a way to evaluate and compare their approaches.

Keywords: Software Architecture Documentation · Natural Language Processing · Traceability link recovery · Mining Software Repositories.

1 Introduction

Benchmarks for evaluation bring numerous benefits (cf. Sim et al. [17]). These benefits include, among others, clearly defined standards and expectations, an increased awareness on related work as well as more frequent collaborations within a domain. However, a recent study by Konersmann et al. [9] shows that between 2017 and 2021 only 2.6% of the 153 full technical papers at the conference-series European Conference on Software Architecture (ECSA) and International Conference on Software Architecture (ICSA) used benchmarks for evaluation. Instead, the most used evaluation strategies are case studies and technical experiments (57%). In order to counteract, the authors propose to mine the public available case studies from the papers to create benchmarks. According to von Kistowski et al. [7], studies can result in such a benchmark. For this purpose, Konersmann et al. [9] provide an overview of different papers that made their case studies publicly available.

Mining existing case studies and example systems to build datasets is a great opportunity for the software engineering domain. The main difficulty is to find such case studies that are reliable and comparable to a specific problem domain. However, it might be hard to find these comparable and exchangeable data. In our case, we look into the well-established research area of traceability link recovery (TLR). TLR approaches create trace links between two or more existing artifacts. These links are particularly helpful for maintenance but help for the overall understanding of a system and how different artifacts and views are connected. However, there are different kinds of artifacts that are traced. Common artifacts are requirement documents, code documents, and issues, but may also include documentation (e.g., textual software architecture documentation) and models. Consequently, many approaches focus on links between requirements and code [1,3,14,18,19], test cases [12], and architecture [11]. Our main focus is recovering trace links between textual software architecture documentation (SAD) and software architecture models (SAM) (cf. [6]). Whenever we mention SAD in the following, we refer to natural language text. SAMs are machine-readable models with an explicit meta-model.

To the best of our knowledge, there is no other work, yet, dealing with links between architecture documentation and models. To allow replicability and to promote comparability, we create a benchmark for traceability link recovery between SAD and SAM. For this, we mine public software repositories, transform the extracted data in unified formats, and label them. The resulting dataset is publicly available [5]. Further, we present a baseline approach to show how the dataset can be used in an evaluation and to provide reference values.

The rest of the paper is structured as follows: We first present the creation of the benchmark dataset (section 2). Following that, we describe its contents in detail (section 3). Lastly, we discuss challenges and opportunities we encountered during creation in section 4.

2 Creating a Benchmark Dataset

Before we created the benchmark dataset, we searched for used datasets in the TLR community. Thereby we found the *CoEST*[1] repository. It currently consists of 15 projects with gold standards for TLR between requirements and source code. However, *CoEST* is not applicable for our work: On the one hand, *CoEST* does not consider architectural descriptions. Further, several projects contain languages other than English. On the other hand, the dataset does not include any architectural models. Although it is possible to synthesize single artifacts by transforming from the other existing artifacts, creating all artifacts synthetically is undesirable. As a result, we can state that we are not aware of a dataset that provides the needed information for TLR between SAD and SAM.

We started to mine open-source software projects for documentations and models to create an initial dataset. In order to find relevant data for our approach, we searched for open-source projects that already contain SADs. We

[1] `http://sarec.nd.edu/coest/datasets.html`

contacted the authors of [2] and retrieved a list of open-source projects that have some architecture documentation from them. In addition, we looked at the repositories of the *Lindholmen dataset* [4]. Even so, we did not find projects with an extensive SAD and a presentation of the architecture (as figure, diagram, or something similar). The lack of architecture documentation in open-source projects is common for small projects. Architecture documents are more often created and maintained in large, successful projects [2].

Since we want the dataset to be heterogeneous, we searched for case studies and example systems of other SAM-based approaches and chose five projects for our initial benchmark dataset:

MediaStore[2]	is a "model application built after the iTunes Store". Its architecture was used for exemplary performance analyses on SAMs.
TEAMMATES[3]	is an open-source "online tool for managing peer evaluations and other feedback paths of your students". TEAMMATES is used as a case study in several SAD-based approaches (cf. [6,15]).
BigBlueButton[4]	is a non-scientific application that provides a web conferencing system with the focus on creating a "global teaching platform".
TeaStore[5]	is a scientific application [8] that is used as a "micro-service reference test application". Like MediaStore, it is used for evaluations of architecture performance analyses.
JabRef[6]	is a tool to manage citations and references in bibliographies. It has features to collect, organize, cite, and share research work.

In order to get more information about the projects used for the benchmark, Table 1 provides a short characterization of them. The table summarizes the main languages (w.r.t. their lines of code), the number of forks, and the amount of contributors. For the MediaStore project, we could not count the number of forks or contributors since it is not published on GitHub.

For each project, we created SAMs for the projects based on either existing models or with the help of the documentation of the projects. We extracted plain-text version from their SADs and created a sentence-wise gold standard for TLR between the SAD and the SAM. More details about the creation of the gold standard and the other artifacts follow in section 3.

The resulting benchmark dataset is described in detail in section 3 and can be found in our public repository [5]. By making the dataset publicly available, we give other researchers the possibility to replicate our results and compare the results of their approaches.

[2] `http://sdq.kastel.kit.edu/wiki/Media_Store`
[3] `http://github.com/TEAMMATES`
[4] `http://bigbluebutton.org`
[5] `http://github.com/DescartesResearch/TeaStore`
[6] `http://github.com/JabRef/jabref`

Project	Languages (kLOC)[7]	Forks	Contributors
MediaStore	Java(4)	N/A	N/A
TEAMMATES	Java(91), TypeScript(54)	$\approx 2.6k$	≈ 500
BigBlueButton	JavaScript(69), JSX(47), Scala(22), Java(21)	$\approx 5.8k$	≈ 180
TeaStore	Java(12)	$\approx 0.1k$	≈ 15
JabRef	Java(157)	$\approx 2.0k$	≈ 490

Table 1. Characteristics of the projects in the benchmark.

3 A Benchmark Dataset for TLR between SAD and SAM

As described in section 2, traceability link recovery benchmarks are very specific regarding their (different) inputs, outputs, and gold standards. Nevertheless, the format of the data should be easily applicable for others. In this section, we provide a closer look on the parts of our dataset and describe our considerations.

3.1 Software Architecture Documentations

Software architecture documentations are one of two input artifacts in our TLR approach. For the benchmark, we obtained the texts of each project by searching their repositories for documentation and looking on their websites for their SAD. Since texts are usually read in and pre-processed with natural language processing tools, we removed tables and figures from the descriptions to provide processable plain text. We additionally cleaned up the texts so that, for example, special characters like curly brackets or captions were removed. For reproduction, we documented all changes made to the texts in the repository. We created for each project a text file containing all sentences of their documentations. In order to simplify the definition of a gold standard for linking sentences and model elements, each line of the file contains exactly one sentence.

In Table 2, we provide insights about the resulting SADs of the projects used for the benchmark dataset, i.e., the number of words and sentences for each SAD. Currently, the shortest SAD of the benchmark consists of 13 sentences (JabRef). The largest includes 198 sentences (TEAMMATES).

3.2 Software Architectural Models

There are different ways to represent software architecture models. MediaStore and TeaStore are systems that we considered for the initial dataset. Since they have already been modeled with the Palladio Component Model (PCM) [13], PCM is our main candidate. Moreover, we chose PCM as meta model because it

[7] rounded kLOC for programming languages with most LOC (calculated via `cloc`)

Project	Words	Sentences
Mediastore	572	37
TeaStore	661	43
TEAMMATES	2509	198
BigBlueButton	1190	85
JabRef	237	13

Table 2. Information about the SADs of the projects in the benchmark.

can cover different views of software architecture (e.g., components and deployment). Thereby, we want to ensure that the benchmark can easily extended. The repository view contains the minimum information to describe the components of a software system. This is enough to run all currently existing approaches. Therefore, when adding further cases to the benchmark that do not provide SAMs as PCM, we only provide the repository view describing the components. If the PCM model of a project already contained more views (e.g., allocation model), we also provide these in our benchmark repository. In general, we plan to add further views to provide more than just the component information of a system. Additionally, we also plan to expand to more model types than just PCM models. In order to increase the benchmark's compatibility with existing approaches, we created UML component models [8] that match the PCM models.

Concluding this section, we summarize information about the different architecture models of the benchmark's projects in Table 3. Since current approaches for TLR between SAD and SAM focus on components, we provide the number of component and number of interfaces of the models of our benchmark. The number of components per project range from 6 to 14 components. Due to the focus on components, the model of JabRef does not contain interfaces.

Project	Components	Interfaces
Mediastore	14	9
TeaStore	11	8
TEAMMATES	8	8
BigBlueButton	12	12
JabRef	6	–

Table 3. Information about the SAMs of the projects in the benchmark.

[8] `http://www.eclipse.org/papyrus/`

3.3 Gold Standard

Besides the input data (SADs and SAMs), the core artifact of our dataset is the manually created gold standard for TLR between SAD and SAM. For each project, the gold standard is available as CSV file. It defines the expected trace links between the sentences of the software architecture documentation and the architectural model elements.

The first project that has been added to our benchmark was TEAMMATES. Its trace links base on a small user study performed as part of a master's thesis [16]. For the other projects, the gold standards have been created separately. To do so, for each project a gold standard has been manually created by one of the authors. Afterwards, the gold standards were analyzed by another author. In the case of different traceability links, the differences were discussed and resolved together. Finally, the gold standard is stored as a CSV file.

To give an example, we consider the eleventh sentence of *MediaStore's* SAD: "The UserManagement component answers the requests for registration and authentication." The repository model of MediaStore contains, among other elements, *Basic Components* that represent components of the system. Since the example sentence mentions the component *UserManagement*, the gold standard contains a link that connects the eleventh sentence referenced by the sentence number (starting at 1) and the component referenced by its unique identifier.

In order to provide more insights about the different projects of the benchmark, we provide numbers for each gold standard in Table 4. We provide the number of trace links, the number of components that have at least one trace link, and the number of sentences that have at least one trace link. Regarding components, we observe that not all components are part of a trace link. Only the SAD of TEAMMATES mentions all components of the model. Additionally, we observe that the share of sentences that contain trace links varies noticeably depending on the project (20% for TEAMMATES and up to 77% for JabRef).

Project	**Trace Links**	**Components with TL**	**Sentences with TL**
Mediastore	29	10	28
TeaStore	27	6	23
TEAMMATES	50	8	39
BigBlueButton	52	11	41
JabRef	18	5	10

Table 4. Information about the gold standards of the projects in the benchmark.

3.4 Simple Tracelink Discovery (STD)

Together with our dataset, we provide Simple Tracelink Discovery (STD)[9], a baseline approach for TLR between SAD and SAM. We provide this approach to provide a simple baseline. Additionally, the approach provides an example on how to use the benchmark and, thus, works as guidance for other researchers.

The main idea of STD is to provide a lower bound. In TLR, it is often assumed that same elements in different artifacts have same names. Thus, a simple base line is to create trace links only when the name of a model element, like a component, is directly mentioned in the text. Therefore, STD matches n-grams of model element names with n-grams of the words within the documentation text. As a result, the precision is usually high while the recall suffers due to the strict assumption. To relax this restriction, there is the option to employ normalized Levenshtein Distance (cf. [10]) to assess name equality, based on a defined threshold. Table 5 displays the evaluation results of STD for the benchmark.

Project	**Precision**	**Recall**	**F1-Score**
Mediastore	1.00	0.62	0.77
TeaStore	0.94	0.57	0.71
TEAMMATES	0.89	0.57	0.70
BigBlueButton	0.88	0.44	0.59
JabRef	0.87	0.42	0.57

Table 5. Evaluation results for the baseline approach STD on the benchmark.

The approach is intended for recovering links between SAD and SAM. Due to its simplicity, it can be easily adapted for other types of artifacts. However, the approach does not perform well if the names of model elements do not appear in both artifacts. For example, if artifacts have a vastly different level of abstraction (e.g., between requirements and code), this baseline approach will most likely perform much worse.

4 Discussion

In this paper, we first discussed the benefits and need for benchmark datasets. Benchmarks enable clear evaluations, comparisons, and provide room for collaborations [17]. Therefore, benchmarks should be established in more research areas. However, lots of research areas are very specific, tailored to particular input and special outputs, and have only small communities. In any of these cases, it is hard to create a dataset or benchmark for the evaluation of an approach. Not only the mining of software repositories and the creation of benchmarks is

[9] `http://github.com/ArDoCo/SimpleTracelinkDiscovery`

a difficult problem, but also the identification of potentially usable data. Therefore, we suggested to extract such data from case studies or examples from other scientific publications relying on the same or very similar inputs. With this data, a dataset with a gold standard can be created.

We followed this idea to create a benchmark dataset with software architectures, texts of software architecture documentation, a gold standard for traceability link recovery, as well as a baseline approach. We also showed the applicability of this process by means of a baseline approach for this traceability link recovery problem.

There are some threats to the validity for the process and our resulting benchmark dataset. First, there may be a threat to validity due to the selected projects. These projects are selected based on literature. All projects have different size and have different architecture styles and patterns. The projects are also from different domains, although they are web-based applications. Additionally, we also assumed similar abstraction levels for SADs and SAMs, which might introduce some bias. Lastly, we created the gold standards ourselves and, thus, might have introduced some bias. We used commonly used techniques like mediation sessions for creating these gold standards, but we cannot rule out that there can still be a certain amount of bias.

In future work, we plan to extend the benchmark dataset with more details for already existing projects. This includes other types of models, more detailed models etc. Moreover, we plan to add more projects to better ensure generalizability of the results when applying the benchmark.

Acknowledgments

This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs. This publication is based on the research project SofDCar, which is funded by the German Federal Ministry for Economic Affairs and Climate Action.

References

1. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering **19**(6), 1565–1616 (2014). https://doi.org/10.1007/s10664-013-9255-y
2. Ding, W., Liang, P., Tang, A., Van Vliet, H., Shahin, M.: How do open source communities document software architecture: An exploratory survey. In: 2014 19th International Conference on Engineering of Complex Computer Systems. pp. 136–145 (2014). https://doi.org/10.1109/ICECCS.2014.26
3. Guo, J., Cheng, J., Cleland-Huang, J.: Semantically enhanced software traceability using deep learning techniques. In: Proceedings of the 39th International Conference on Software Engineering. p. 3–14. ICSE '17, IEEE Press (2017). https://doi.org/10.1109/ICSE.2017.9

4. Hebig, R., Quang, T.H., Chaudron, M.R.V., Robles, G., Fernandez, M.A.: The quest for open source projects that use uml: Mining github. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. p. 173–183. MODELS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976767.2976778
5. Keim, J., Fuchß, D., Corallo, S.: Architecture Documentation Consistency Benchmark (2022). https://doi.org/10.5281/zenodo.6966831, `https://github.com/ArDoCo/Benchmark`
6. Keim, J., Schulz, S., Fuchß, D., Kocher, C., Speit, J., Koziolek, A.: Trace link recovery for software architecture documentation. In: Biffl, S., Navarro, E., Löwe, W., Sirjani, M., Mirandola, R., Weyns, D. (eds.) Software Architecture. pp. 101–116. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-86044-8_7
7. v. Kistowski, J., Arnold, J.A., Huppler, K., Lange, K.D., Henning, J.L., Cao, P.: How to build a benchmark. In: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering. p. 333–336. ICPE '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2668930.2688819
8. von Kistowski, J., Eismann, S., Schmitt, N., Bauer, A., Grohmann, J., Kounev, S.: TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In: Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems. MASCOTS '18 (September 2018). https://doi.org/10.1109/MASCOTS.2018.00030
9. Konersmann, M., Kaplan, A., Kühn, T., Heinrich, R., Koziolek, A., Reussner, R., Jürjens, J., al Doori, M., Boltz, N., Ehl, M., Fuchß, D., Großer, K., Hahner, S., Keim, J., Lohr, M., Sağlam, T., Schulz, S., Töberg, J.P.: Evaluation methods and replicability of software architecture research objects. In: 2022 IEEE 19th International Conference on Software Architecture (ICSA). pp. 157–168 (2022). https://doi.org/10.1109/ICSA53651.2022.00023
10. Levenshtein, V.I., et al.: Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics doklady. vol. 10, pp. 707–710. Soviet Union (1966)
11. Molenaar, S., Spijkman, T., Dalpiaz, F., Brinkkemper, S.: Explicit alignment of requirements and architecture in agile development. In: Madhavji, N., Pasquale, L., Ferrari, A., Gnesi, S. (eds.) Requirements Engineering: Foundation for Software Quality. pp. 169–185. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-44429-7_13
12. Rempel, P., Mäder, P.: Estimating the implementation risk of requirements in agile software development projects with traceability metrics. In: Fricker, S.A., Schneider, K. (eds.) Requirements Engineering: Foundation for Software Quality. pp. 81–97. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-16101-3_6
13. Reussner, R., Becker, S., Burger, E., Happe, J., Hauck, M., Koziolek, A., Koziolek, H., Krogmann, K., Kuperberg, M.: The palladio component model. Tech. Rep. 14, Karlsruher Institut für Technologie (KIT) (2011). https://doi.org/10.5445/IR/1000022503
14. Rodriguez, D.V., Carver, D.L.: Multi-objective information retrieval-based NSGA-II optimization for requirements traceability recovery. In: 2020 IEEE International Conference on Electro Information Technology (EIT). pp. 271–280 (2020). https://doi.org/10.1109/EIT48999.2020.9208233, ISSN: 2154-0373

15. Schröder, S., Riebisch, M.: An ontology-based approach for documenting and validating architecture rules. In: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings. ECSA '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3241403.3241457, https://doi.org/10.1145/3241403.3241457
16. Schulz, S.: Linking Software Architecture Documentation and Models. Master's thesis, Karlsruher Institut für Technologie (KIT) (2020). https://doi.org/10.5445/IR/1000126194
17. Sim, S.E., Easterbrook, S., Holt, R.C.: Using benchmarking to advance research: A challenge to software engineering. In: Proceedings of the 25th International Conference on Software Engineering. p. 74–83. ICSE '03, IEEE Computer Society, USA (2003)
18. Wang, W., Niu, N., Liu, H., Niu, Z.: Enhancing automated requirements traceability by resolving polysemy. In: 2018 IEEE 26th International Requirements Engineering Conference. pp. 40–51 (2018). https://doi.org/10.1109/RE.2018.00-53
19. Zhang, Y., Wan, C., Jin, B.: An empirical study on recovering requirement-to-code links. In: 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). pp. 121–126 (2016). https://doi.org/10.1109/SNPD.2016.7515889