Ideas on Improving Software Artifact Reuse via Traceability and Self-Awareness

Christof Tinnes*, Andreas Biesdorf†, Uwe Hohenstein† and Florian Matthes*
* *Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany*
{*christof.tinnes,matthes*}*@tum.de*
†*Siemens AG - Corporate Technology, Otto-Hahn-Ring 6, München 81739, Germany*
{*andreas.biesdorf,uwe.hohenstein*}*@siemens.com*

***Abstract*—We describe our vision towards automatic software and system development and argue that reusing knowledge from existing projects as well as traceability between corresponding artifacts are important steps towards this vision. We furthermore list barriers that are currently experienced with software artifact reuse and traceability in industry, and suggest some ideas to overcome these barriers.**

1. Introduction

Imagine a sociotechnical system which produces software out of high-level goals or strategies. Without adding further constraints, we could easily think of such a system. If, for example, the system includes humans and human actions, the system could be a software company. If we add the constraint that the system should not involve any human activities to achieve its goals, such a system does not exist given that the high-level goals are non-trivial. Still, systems that can produce software systems out of high-level goals are either involving high amount of human work or the software systems produced are very limited in their capabilities or their variability (e.g., software product lines). ISO/IEC/IEEE 12207:2017 [15] describes processes in the software development life cycle. Most of the processes described in this standard, such as the requirements analysis process, the architectural design process or the implementation process, still include a lot of manual and human efforts.

We believe that the current state of technologies allows for a higher level automation in the software development life cycle. Moreover, reuse of software artifacts (not only source code but also design artifacts, operational artifacts and all other data and information that is created during the software development life cycle) combined with traceability between artifacts, may provide the basis for replacing human activities by automated activities.

In this paper, we discuss current barriers to an efficient application of automated or semi-automated reuse in the development of large-scale software-intensive systems in an industrial setting. Furthermore, we list some ideas to overcome these barriers. We propose a high-level conceptual reference model for a reuse recommender system and argue that the viewpoint of Self-Aware Computing Systems [16] is beneficial to study, discuss and build such recommender systems.

2. The Need for Reuse and Traceability

When making a decision, humans usually rely on existing experience they have gained in the past. Humans abstract the decision making task at hand and compare it to "similar tasks" from the past. The decisions made and their outcomes are then used to make a decision in the current decision making task [3, 21]. When moving towards the long-term vision of fully automated software system development, we think that it is necessary to find similar tasks and decisions of existing projects and subsequently evaluate their outcome. The software artifacts themselves or meta-information from existing projects as well as from the current project must be available. In summary, we are looking for a computing system that is

- capturing knowledge (e.g., requirements, design decisions, source code, context information) about itself and its environment,
- learning models about itself (e.g., to determine what a "similar task" means), and
- reasoning (make informed decisions using previous outcomes) to act (execute the decisions) based on these models.

Computing systems with these capabilities have recently been defined as *Self-Aware Computing Systems* [16]. We will focus here on how to determine "similar tasks" or requirements. The "acting" part would then include adjustments and reuse of previous artifacts, as for example, existing source code. In many cases, it might not be possible to completely automate this acting part but to provide recommendations instead, which support reuse. To reuse artifacts based on similar requirements, we need traceability between the requirements and other artifacts [5, 20]. To summarize, we argue that reuse of software artifacts and traceability between artifacts is an important step towards a computing system that is able to take high-level goals into account and towards moving from human activities in software development to computing activities.

3. Barriers to Reuse and Traceability

Our first goal is to build a system that is able to make automated recommendations for software artifact reuse based on existing knowledge gained from other projects. Especially,

we are interested in reusing code, design decisions or third-party libraries based on given requirements or user stories. We are aware that there have already been several attempts to combine reuse of high-level software development artifacts and traceability [5, 14, 20, 22], but we do not know of any automatic approach that is efficiently used in the development of large-scale software-intensive systems in an industrial setting.

We conducted a workshop with five system architects and two user experience experts to determine barriers to effective reuse in software development. As a result of this workshop, we identified *four high-level aspects* that need to be considered when designing an effective support system for software artifact reuse:

- *Human aspects* - According to one of the system architects, architects and developers tend to reject existing solutions ("There is already a solution to the problem, but I do not like it!"). Instead of using or adjusting an existing solution, software systems are often re-invented from scratch. This observation is quite common in software engineering and referred to as Not-Invented-Here syndrome [1].
- *User experience aspects* - User experience experts emphasized that users will more easily adopt a solution, if "it is fun".
- *Technical aspects* - Reuse of complex software artifacts requires finding and understanding existing artifacts. In typical industrial settings, the quality of the artifacts and information available for artifacts varies widely. Similarity detection and traceability are therefore no easy tasks.
- *Organizational aspects* - Obviously, artifacts of projects need to be available and accessible, which is often not the case.

By "solution", we mean software artifacts that serve a given purpose. E.g., a design specification can be consistent to some given requirements and therefore is a solution to the requirements. Source code can implement this specification and therefore is a solution to it.

In the following, we will focus on the technical aspects. We believe that previous approaches suffer from the following shortcomings:

1) There are very promising results in traceability research. In [11] word embeddings and deep learning are successfully applied to software traceability. Current research concentrates on specific aspects of reuse or traceability. However, to the best of our knowledge, there has not yet been the attempt to design and build an integrated system that combines multiple approaches and is able to adjust and reuse the models from one domain to another domain.
2) Even though there are excellent tools for traceability available (e.g., TraceLab[1]), these tools are rarely capable of being integrated into the existing day-to-day activities and tools the users are familiar with. Most of the tools are provided for research purposes and not for industrial application.
3) Most of the approaches did not have access to very powerful NLP techniques that exist today. For example, there are techniques available that capture the semantics of words and documents and can be trained fast and unsupervised on existing corpora [4, 18, 19].
4) Continuous learning through user feedback and continuous adaption of the used techniques have to be in place. For example, the combination of techniques that are used for a given task, e.g., document similarity, need to be adapted continuously based on the existing data and user feedback. This allows to "finetune" pretrained models and adapt them to the specific project needs.

1. https://github.com/CoEST/TraceLab

In the following, we describe our planned approach to overcome these four barriers.

4. Approach

4.1. Conceptual Reference Model

During the software development life-cycle, various artifacts are produced and the dependencies among them can be very complex. There are frameworks available, e.g., the TOGAF Architecture Content Framework [10], which aim at structuring the output in different life-cycle phases. As an example of the complexity, artifacts are usually not in a one-to-one relation but in a many-to-many relation, and the relations are changing over time. For example, a line of code is usually not only related to one user story, but might cover multiple user stories (the line of code might contain an if-statement, which distinguishes both stories). Some of the data is structured, some is unstructured, and some of the data is only partially structured. E.g., issues in an Issue Management System (IMS) have a field owner so the linking of issues and issue owner is directly accessible. On the other hand, the description of an issue usually consists of natural language and therefore is unstructured. Commit messages, for example, sometimes follow a template including corresponding IMS issue id and sometimes are completely unstructured. In order

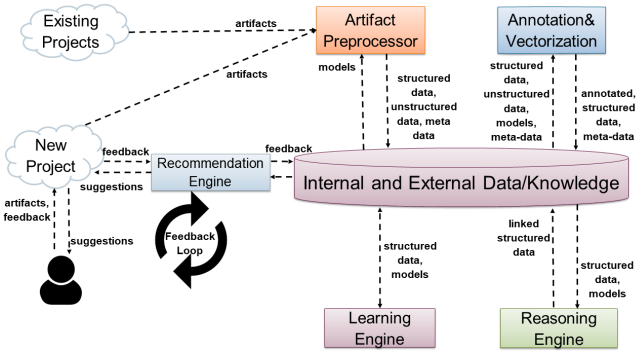

Figure 1. A conceptual reference model for a software development artifact recommender system.

to use the unstructured data for making recommendations, a lot of preprocessing has to be in place. When some information has been extracted, it needs to be linked and used for reasoning. For instance, trace links between the artifacts can be computed. Usually, some pipelines are proposed to process given artifacts [17]. This makes sense for a concrete task (e.g., generating trace links between test case specification and source code), but given the complexity described above, the management of pipelines for all single tasks that occur in the overall picture will become too complex. We argue that a blackboard architecture [2] is much better suited, to handle the complexity. In this architecture, the existing knowledge and new artifacts are stored in a shared data repository. There are agents for specific tasks, e.g., preprocessing of textual data or annotation of data that process the data from this repository. The agents are pluggable so that the systems' capabilities can be extended by adding more agents. The conceptual model for such a system is depicted in Figure 1. The system typically needs agents to preprocess natural language texts, compute similarities of artifacts, annotate documents with topics or architectural elements and compute trace links. Besides the agents that (pre-)process and enrich the available knowledge, this model includes a recommendation engine and a learning engine, which are described in the next subsection.

4.2. Continuous Learning

In a first step, the system we envision makes recommendations to users. For example, if a user creates a new issue in an IMS, the recommendation engine finds similar issues that are already available (from other inner-source or open-source projects). The user can then reuse (parts of) the solution for these issues by utilizing trace links between the issues and solution artifacts. User feedback can be utilized to adjust the models that are used by the recommendation engine. The recommendations are usually influenced by a large number of different factors, such as parameters of the preprocessing, filters, model parameters, ranking criteria and the models themselves. One challenge therefore is to find the best values for the influencing factors. Initial values for these factors can be learned from existing labeled data. Usually, there will not be sufficient learning data and furthermore, the values will change over time and need to be adjusted to the given context. Therefore, our conceptual model includes a learning engine. This engine evaluates user feedback, which is extracted from the projects (e.g., rating system integrated into an IMS). The engine can use existing technologies, e.g., from learning to rank (LTR) algorithms [24], to improve the recommendations based on the user feedback.

4.3. Self-Aware Computing Systems

Comparing the conceptual model of Figure 1 and the reference architecture for Self-Aware Computing Systems given in the first chapter of [16], our conceptual model follows the so called LRA-M (**L**earn-**R**eason-**A**ct and **M**odel) reference architecture. We believe that the viewpoint of Self-Aware Computing Systems can be fruitful in software and systems traceability. A model-based learning and reasoning feedback loop can be used to improve the factors and the models that influence the recommendations as described in Subsection 4.2. It therefore helps to overcome the complexity of the problem.

4.4. Integration into the Tool Landscape

There are already many tools for requirements and software traceability (e.g., TraceLab[2], OpenTrace[3]). Even though these are excellent tools, they are often made for research purposes and can hardly be applied for software artifact reuse. In-practise tools usually need to be explicitly integrated into the existing tool landscape and development activities. Since this leads to additional usage overhead and learning efforts (depending on the existing tool landscape), acceptance of such tools is often limited. On the other hand, applications that are already used for requirements management (e.g., IBM DOORS[4]), often lack capabilities for semi-automated trace link generation and usually cover only parts of the development life-cycle. As can be seen from our conceptual model, our approach is to feed the recommendations back "into the projects". To this end, the software that is used in day-to-day activities such as Integrated Development Environments, Modelling Tools and IMS needs to be integrated with such a tool for requirements and software traceability. We therefore envision an open system. Integration with existing tools can be done, for example, by means of providing plugins.

4.5. The Lexical Gap, Natural Language Semantics and Artificial Intelligence

Artifacts produced during system and software development differ along multiple dimensions. They vary from formal language to natural language, structured to unstructured, text to pictures, in file type, language, etc. This makes especially the generation of trace links between artifacts a very complex task. Recent developments in Natural Language Processing (NLP) though seem to be quite promising to overcome these challenges [23]. Word2Vec [18] for example, allows to train vector representations on large corpora with a low computational complexity and provides good results, e.g., in capturing semantic similarity of words. These techniques could even be used to overcome the lexical gap between natural language text and source code as recent studies demonstrate [23], and therefore allow to compute trace links between the artifacts. Other research areas in Artificial Intelligence (AI) also present promising results in software engineering. For instance, Search-Based-Software-Engineering [12] is successfully applied to the recovery of trace links [8].

5. Relevance

The relevance of reusing artifacts that are created during the software and systems development lifecycle has often been stressed in the literature. Unlike for software components or libraries, the reuse of knowledge and experiences from

2. https://github.com/CoEST/TraceLab
3. http://www.semanticsoftware.info/opentrace
4. https://www.ibm.com/de-de/marketplace/requirements-management

other projects as well as the consideration of high-level artifacts is less mature, although its importance has been emphasized many times. For example, in [13], the authors come to the conclusion that software architects see the importance of documenting architectural knowledge but are rather reluctant in actively documenting their own knowledge. Actively maintaining trace links is not part of the work habits of software architects. A study on the effect of using requirements catalogs on effectiveness and productivity of requirements specification [7] concludes that catalog-based reuse has a large effect on effectiveness. An industrial case study by Goldin et al. [9] demonstrates the reduction of time to market by reusing requirements. According to the study by Falessi et al. [6] using NLP techniques can lead to an effort reduction of up to 12% for classifying equivalent requirements.

During the workshop mentioned in Section 3, the need for an integrated end-to-end support for reuse of existing software artifacts has been stressed. This furthermore encourages us to foster the vision and pursue research in accordance with our approach described previously.

6. Conclusion and Outlook

The idea behind Self-Adaptive or Self-Aware Computing Systems is that of deferring binding time, i.e. moving activities from planning time, design time or implementation time to runtime. The purpose of software systems is to support human activities by computing systems, i.e., also moving activities to runtime. The planning, development and operations of these software systems themselves includes human activities, some of which have already been automated (e.g., monitoring and restarting applications). We argue that as a next step towards our vision it is necessary to reuse existing knowledge and existing artifacts. In order to enable computing systems to learn from the experiences of previous projects and use existing knowledge for decisions, the system needs to be able to understand and compare the context of different projects. We believe, that cross-project reuse enabled by traceability is a first step into that direction. Moreover, we believe that the shortcomings of existing systems for reuse and traceability can be overcome. Appropriate systems, fit for industrial usage, can be created with the current state of technology. In the future, this approach can be utilized to support complex decision-making tasks, e.g., by applying reinforcement learning techniques. Furthermore, using the trace links, available temporal structures can be "lifted between artifacts". For instance, using links between source code commits and issues in an IMS, the commit history can be projected to the issues and therefore creating a history of the tickets. This is the point, where the potential of the reuse of high-level requirements becomes enormous. Not only source code can be reused but one can also anticipate bugs, changes or possible design flaws since they are available in the history of other projects. We believe that this step can be reached in a reasonable amount of time and that all the ingredients needed for this step are already there. They just need to be assembled in the right way.

References

[1] Dan Ariely. *The Upside of Irrationality: The Unexpected Benefits of Defying Logic*. Harper Perennial, 1st edition, 2011.

[2] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996.

[3] P. Dayan and N. D. Daw. Decision theory, reinforcement learning, and the brain. *Cogn., Affect. Behavioral Neurosci.*, 8(4):429–453, 2008.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Randa Elamin and Rasha Osman. Towards Requirements Reuse by Implementing Traceability in Agile Development. *41st Annual Computer Software and Applications Conference*, 2017.

[6] Davide Falessi and Giovanni Cantone. The Effort Savings of Using NLP Techniques for Classifying Equivalent Requirements in Industry. *IEEE Software*, In Print(c):1, 2018.

[7] J. L. Fernández-alemán, J. M. Carrillo-de gea, J. V. Meca, J. N. Ros, A. Toval, and A. Idri. Effects of Using Requirements Catalogs on Effectiveness and Productivity of Requirements Specification in a Software Project Management Course. *IEEE Trans. Edu.*, 59:105–118, 2016.

[8] A. Ghannem and M. S. Hamdi. Search-Based Requirements Traceability Recovery. *IEEE Congr. on Evol. Comp.*, 3:1183–1190, 2017.

[9] Leah Goldin, Michal Matalon-beck, and Judith Lapid-maoz. Reuse of Requirements Reduces Time to Market. *IEEE Int. Conf. on Soft. Sci., Techno. & Eng.*, pages 55–60, 2010.

[10] The Open Group. *TOGAF Version 9.1*, 2011.

[11] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. Semantically Enhanced Software Traceability Using Deep Learning Techniques. *Proc. - ICSE 2017*, pages 3–14, 2017.

[12] M. Harman. The role of artificial intelligence in information retrieval. *Proc. 1st Int. Workshop Realizing AI Synergies in Softw. Eng.*, 2012.

[13] Johan F. Hoorn, Rik Farenhorst, Patricia Lago, and Hans Van Vliet. The lonesome architect. *J. Syst. Softw.*, 84(9):1424–1435, 2011.

[14] B. Imam, A. Nordin, and N. Salleh. Software Requirements Patterns and Meta model : A Strategy for Enhancing Requirements Reuse (RR). *Int. Conf. Inform. Commun. Technol. Muslim World*, 2016.

[15] ISO, IEC, and IEEE. Systems and software engineering. Technical report, ISO/IEC/IEEE 12207, 2017.

[16] S. Kounev, P. Lewis, K. L. Bellman, N. Bencomo, J. Camara, A. Diaconescu, L. Esterle, K. Geihs, H. Giese, S. Gtz, P. Inverardi, J. O. Kephart, and A. Zisman. *Self-Aware Computing Systems*. Springer, 2017.

[17] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proc. of 52nd Annu. Meeting of the Assoc. for Computational Linguistics: Sys. Demonstrations*, pages 55–60. Association for Computational Linguistics, 2014.

[18] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *NIPS proceedings*, 2013.

[19] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[20] Rob Pooley and Craig Warren. Reuse through requirements traceability. *Proceedings - The 3rd International Conference on Software Engineering Advances, ICSEA 2008, Includes ENTISY 2008: International Workshop on Enterprise Information Systems*, pages 65–70, 2008.

[21] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[22] Tuyet-lan Tran and Joseph S. Sherif. Quality Function Deployment (QFD): An Effective Technique For Requirements Acquisition and Reuse. *Proc. of Softw. Eng. Standards Symp.*, 1995.

[23] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu. From word embeddings to document similarities for improved information retrieval in software engineering. *Proc. 38th Int. Conf. Softw. Eng. - ICSE '16*, pages 404–415, 2016.

[24] T. Zhao, Q. Cao, and Q. Sun. An Improved Approach to Traceability Recovery Based on Word Embeddings. *Proc. - Asia-Pacific Soft. Eng. Con.*, 2017-Decem:81–89, 2018.