Cost-Effective Traceability Links for Architecture-Level Software Understanding: A Controlled Experiment

Muhammad Atif Javed, Srdjan Stevanetic and Uwe Zdun
Software Architecture Research Group
University of Vienna, Austria
muhammad.atif.javed|srdjan.stevanetic|uwe.zdun@univie.ac.at

ABSTRACT

An important architectural challenge is to recover traceability links between the software architecture and artifacts produced in the other activities of the development process, such as requirements, detailed design, architectural knowledge, and implementation. This is challenging because, on the one hand, it is desirable to recover traceability links of a high quality and at the right quantity for aiding the software architect or developer, but, on the other hand, the costs and efforts spent for recovering should be as low as possible. The literature suggests manual, semi-automatic, and automatic recovery methods, each of which exhibits different impacts on costs as well as quantity and quality of the recovered links. To date, however, none of the published empirical studies have comparatively examined the automation alternatives of traceability link recovery. This paper reports on a controlled experiment that was conducted to investigate how well typical results produced by the three automation alternatives support human software developers in architecture-level understanding of the software system. The results provide statistical evidence that a focus on automated information retrieval (IR) based traceability recovery methods significantly reduces the quantity and quality of the elements retrieved by the software developers, whereas no significant differences between manual and semi-automatic traceability link recovery were found.

CCS Concepts

•Software and its engineering → Software architectures; Documentation; Empirical software validation;

Keywords

Traceability, Software architecture, Automation alternatives, Empirical Software Engineering, Controlled experiment

ASWEC '15, September 28 – October 01, 2015, Adelaide, SA, Australia

DOI: 10.1145/1235

1. INTRODUCTION

Manual traceability recovery requires human effort for identification of traceability links, in which a quantity of 100% correct and also 100% actually correct links is achievable. Semi-automatic traceability recovery involves automatic methods together with human activities, in which in most of the cases a quantity of 10–35% correct links is achievable at actually correct link levels of 80–100% [2][4]. Automatic traceability recovery is based on automatic methods (without human involvement), in which in most of the cases a quantity of 90% correct links is achievable at actually correct link levels of 5–30% [12][14]. So far, however, none of the published empirical studies have compared these automation alternatives for the recovery of traceability links.

The goal of this paper is to perform a pair-wise comparison of the automation alternatives of traceability link recovery in terms of their impact (low or high) on quantity and quality of retrieved elements during architecture-level understanding of the software system. Specifically, we intend to answer the following research question: Are the quantity and quality of retrieved elements achievable with semi-automatic traceability link recovery (i.e., low quantity but high quality links) better than those achievable with automatic traceability link recovery (i.e., high quantity but low quality links) and reasonably close to results achievable with manual traceability link recovery (i.e., the entire links)? To answer the research question, we conducted a controlled experiment at the University of Vienna, Austria, in January 2015. The experiment was announced as a practical session on software architecture understanding. In total, 74 students of the Information and Software Technology course took part.

The participants were asked to perform eight activities aimed at gaining an architecture-level understanding of the Apache Wink Version 1.4.0, a framework for the development and consumption of REpresentational State Transfer (REST) based web services. In the experiment, the participants were assigned to three balanced groups, referred to as manual, semi-automatic and automatic groups. Each of the groups received a set of traceability links created by either a manual, semi-automatic and automatic, respectively, and we checked that the quantity and quality of recovered links in the groups were representative for manual, semi-automatic and automatic traceability tools (based on the data reported in the literature). The data from the experiment was analysed, and the quantity and quality of retrieved elements by the different groups were compared. The results of the experiment demonstrate that automatic traceability recovery

significantly reduces the quantity and quality of retrieved elements, whereas no significant differences between manual and semi-automatic traceability recovery were found.

The rest of this paper is organized as follows: Section 2 discusses the related work. Section 3 describes the design of the controlled experiment including the introduction of variables and hypotheses. Section 4 presents the hypotheses tested and the analysis of the results of the study. Section 5 discusses threats to validity. Section 6 concludes the study and discusses future work.

2. RELATED WORK

A few empirical studies have been performed to evaluate the added value of traceability links. In our earlier work [6][5][7], we analysed the support provided by manual traceability. Our findings show that the use of traceability links significantly increases the correctness of the answers of the participants, whereas no conclusive evidence concerning the influence of the experience of the participants was observed. The results also demonstrate that using traceability links leads to slight difference in the quantity and quality of recovered elements for a larger software system.

Cuddeback et al. [1] and Dekhtyar et al. [3] investigate the usefulness of information retrieval (IR) based traceability recovery tools. Cuddeback et al. findings show that the participants failed to finalize the correct traceability links, while the participants provided with the lower recall and precision of traceability links make significant improvements. In addition, regardless of size and accuracy of the initial traceability links, the participants tend to guess the correct number of traceability links. Dekhtyar et al. performed two more follow-up experiments. The results demonstrate that the accuracy of initial traceability links and time spent had significant interaction with the final traceability links, whereas no significant differences with regard to the tool used, effort applied in searching for missing links and traceability experience were observed.

The contribution of this study is novel for three main reasons. First, none of the published empirical studies have examined the added value of semi-automatic traceability recovery. Second, there exist no comparative evidence on how well each automation alternative supports the quantity and quality of retrieved elements. Third, the earlier works on human analyst effort and performance are based on requirement traceability links, while this experiment investigates the support provided by architecture traceability links.

3. DESIGN OF THE EXPERIMENT

For the study design, the guidelines for controlled experiments by Kitchenham et al. [8] and Wohlin et al. [13] were used. The former guidelines were primarily used in the planning phase of our experiment, while the latter was used as a reference for the analysis and interpretation of the results.

3.1 Goal, hypotheses, parameters, and variables

The goal of the experiment is to empirically investigate in how far the use of different automation alternatives for traceability link recovery supports architecture-level understanding of the software system. The experiment goal led to the following null hypotheses and corresponding alternative hypotheses:

H_{01}: Traceability links produced by an automated method **lead to a higher quantity of correctly retrieved elements** during architecture-level understanding of the software system than traceability links produced by the manual and semi-automatic methods.
H_1: Traceability links produced by an automated method **lead to a lower quantity of correctly retrieved elements** during architecture-level understanding of the software system than traceability links produced by the manual and semi-automatic methods.

H_{02}: Traceability links produced by an automated method **lead to a lower quantity of incorrectly retrieved elements** during architecture-level understanding of the software system than traceability links produced by the manual and semi-automatic methods.
H_2: Traceability links produced by an automated method **lead to a higher quantity of incorrectly retrieved elements** during architecture-level understanding of the software system than traceability links produced by the manual and semi-automatic methods.

H_{03}: Traceability links produced by an automated method **lead to a higher overall quality of retrieved elements** during architecture-level understanding of the software system than traceability links produced by the manual and semi-automatic methods.
H_3: Traceability links produced by an automated method **lead to a lower overall quality of retrieved elements** during architecture-level understanding of the software system than traceability links produced by the manual and semi-automatic methods.

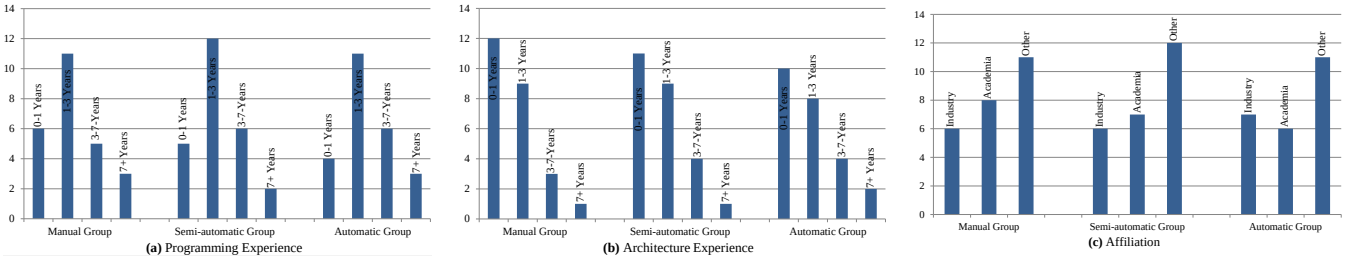
3.1.1 Dependent and independent variables

Nine dependent and five independent variables were observed during the experiment. The quantity of correctly and incorrectly retrieved elements, and their overall quality for the manual, semi-automatic and automatic traceability link recovery are dependent variables. They were calculated by using the standard information retrieval metrics, in particular, recall, precision, and f-measure, respectively. Recall is the percentage of correct matches retrieved by an experiment subject, while precision is the percentage of retrieved matches that are actually correct. Because recall and precision measure two different concepts, it can be difficult to balance between them. Therefore, f-measure, a standard combination of recall and precision, defined as their harmonic mean, is used to measure the overall quality of retrieved elements from the experiments' participants.

The independent variables relate to the personal information (programming experience, architecture experience, affiliation), group affiliation (manual group, semi-automatic group or automatic group) and time spent in the experiment. These variables could have an influence on the dependent variables, which is eliminated by balancing the characteristics between the manual group, semi-automatic group and the automatic group.

3.2 Experiment Design

To test the hypotheses, we conducted a controlled experiment at the University of Vienna, Austria. The experiment was conducted as practical session on software architecture understanding.



(a) Programming Experience **(b)** Architecture Experience **(c)** Affiliation

Figure 1: Distribution of Participants

3.2.1 Participants

The participants in the experiment were 74 students of the Information and Software Technology course held at University of Vienna. Twenty-five individual students took part in the manual and semi-automatic groups, while the other twenty-four students have participated in the automatic group.

3.2.2 Objects

The software system to be architecturally understood by participants was the Apache Wink[1] Version 1.4.0. It is a framework for the development and consumption of REST based web services.

3.2.3 Blocking

To be able to explicitly analyse the influence of automation alternatives on the quantity and quality of retrieved elements, the participants (also referred to as human analysts) were randomly assigned to the three balanced groups. Figure 1 shows the distribution of the participants based on their previous experience and affiliation, as assigned to the manual group, semi-automatic group and the automatic group. The Sub-figures (a) and (b) show the previous experience of the participants concerning programming and software architecture, while Sub-figure (c) shows the affiliation of the participants. Note that the overall experiences and affiliations are rather well balanced in the experiment.

3.2.4 Instrumentation

The instruments discussed in the following paragraphs were used to carry out the experiment.

Documentation about the Apache Wink Version 1.4.0: The documentation describes the high-level conceptual features and lists technologies and frameworks used in the implementation. Besides text, a UML component diagram is used to illustrate the components, and their inter-relationships in parts of the architecture.

Access for the manually recovered traceability links: The participants in the manual group were provided with the manually identified traceability links, in which a quantity of 100% correctly elements is likely achieved at an actually correct link level of 100%. The participants in the manual group were explicitly told that they received the entire traceability links for Apache Wink Version 1.4.0.

Access for the semi-automatically recovered traceability links: The participants in the semi-automatic group were provided with the semi-automatically generated traceability links. In particular, these links were generated based on the traceability rules, in which a quantity of 15.6% correct elements is achieved at an actually correct link level of 100%. The participants in the semi-automatic group were explicitly told that they received the rule-based generated links for Apache Wink Version 1.4.0 that are incomplete. Therefore, they needed to find the remaining set of relevant classes (e.g., by exploring the imports and source code packages of the listed classes).

Access for the automatically recovered traceability links: The participants in the automatic group were provided with links generated using a state-of-the-art information retrieval (IR) methods based tool. These links were generated using the Traceclipse tool [9]. Due to the fact that the current IR-based traceability tools result in a low precision and proliferation of traceability links, we decided to keep the most important generated links. After deeper exploration, it was perceived that the first 600 links (with similarity threshold > 1.5) cover most of the important links for Apache Wink Version 1.4.0, and those were therefore selected for the experiment. The participants in the automatic group were explicitly told that they received generated links. Therefore, they need to study the source code as well, as IR-based links might be misleading (i.e., incorrect or incomplete).

A questionnaire to be filled-in by the experiments' participants: At the first page of the questionnaire, the participants had to rate their programming experience, architecture experience and affiliation, while the subsequent pages contains the eight understanding activities. The activities highlight many of the Apache Wink aspects at both high-level and low-level of abstraction.

4. ANALYSIS

4.1 Quantity of correctly retrieved elements

To be able to test the first null hypothesis H_{01}, the influence of three automation alternatives of traceability link recovery on the quantity of correctly retrieved elements is measured. In the analysis of the experiment, the Kruskal-Wallis test [10] and pairwise Wilcoxon Rank-Sum test [11] are used. First, the Kruskal-Wallis test is used to find out whether a significant difference exists between the participant groups. Second, the corresponding post-hoc test, pairwise Wilcoxon Rank-Sum test, is used to perform the individual comparisons between the participant groups.

Table 1 shows the results of the pairwise Wilcoxon ranksum test for the manual, semi-automatic and automatic groups.

[1]https://wink.apache.org/

The table shows that the experiment provides strong evidence that H_{01} can be rejected. This means that in our experiment the links produced by an automated traceability method lead to a lower quantity of correctly retrieved elements during architecture-level understanding of the software system than traceability links produced by the manual and semi-automatic methods.

Factor	Pairwise Wilcoxon Rank-Sum Test
Manual Group vs. Semi-Automatic Group	p-value = 0.109430
Semi-automatic Group vs. Automatic Group	**p-value = 0.000970**
Manual Group vs. Automatic Group	**p-value = 0.000077**

Table 1: Post-hoc pairwise comparisons (after Kruskal-Wallis test) for quantity of correctly selected elements

4.2 Quantity of incorrectly retrieved elements

Hypothesis H_{02} was also evaluated with a pairwise Wilcoxon rank-sum test after a Kruskal-Wallis test that indicated a significant difference between the participant groups. The results are shown in Table 2. The table shows that the experiment provides strong evidence that H_{02} can be rejected. This means that in our experiment the links produced by an automated traceability method lead to a higher quantity of incorrectly retrieved elements during architecture-level understanding of the software system than traceability links produced by the manual and semi-automatic methods.

Factor	Pairwise Wilcoxon Rank-Sum Test
Manual Group vs. Semi-Automatic Group	p-value = 0.4000000
Semi-automatic Group vs. Automatic Group	p-value = **0.0000130**
Manual Group vs. Automatic Group	p-value = **0.0000099**

Table 2: Post-hoc pairwise comparisons (after Kruskal-Wallis test) for quantity of incorrectly selected elements

4.3 Overall quality of retrieved elements

The pairwise Wilcoxon rank-sum test is also used to evaluate the Hypothesis H_{03} after a Kruskal-Wallis test that indicated a significant difference between the participant groups. The results are shown in Table 3. The table shows that the experiment provide strong evidence that H_{03} can be rejected. This means that in our experiment the links produced by an automated traceability method lead to a lower overall quality of retrieved elements during architecture-level understanding of the software system than traceability links produced by the manual and semiautomatic methods.

Factor	Pairwise Wilcoxon Rank-Sum Test
Manual Group vs. Semi-Automatic Group	p-value = 0.220000000
Semi-automatic Group vs. Automatic Group	p-value = **0.000000110**
Manual Group vs. Automatic Group	p-value = **0.000000096**

Table 3: Post-hoc pairwise comparisons (after Kruskal-Wallis test) for overall quality of selected elements

5. THREADS TO VALIDITY AND LIMITATIONS OF THE STUDY

Multiple levels of validity threats have to be considered in the experiment. The internal validity refers to the cause effect inferences between the treatment and the dependent variables measured in an experiment. External validity concerns the generalizability of the results for a larger population. Construct validity focuses on the suitability of the study design for the theory behind the experiment. Finally, conclusion validity focuses on the relationship between treatment and outcome and on the ability to draw conclusions from this relationship.

Internal validity. The experiment was conducted in a controlled environment in separate rooms under supervision of at least one experimenter. Although, it is not possible to completely prohibit misbehaviour or interaction among participants, it is not very likely that misbehaviour or interactions have had a big influence on the outcomes of the experiment.

Another potential threat to validity is that the analysts could have been biased towards a specific group. We tried to exclude this threat to validity by not revealing the identity of the participants or in which of the three groups they have participated to the analysts. Hence, it is rather unlikely that this threat occurred.

External validity. As discussed in Section 3.2, the experiment was conducted with 74 students of the Information and Software Technology course. Nevertheless, the results of our previous study, where we compared the results from two controlled experiments with students and professionals, imply that the participants' experience does not have a significant influence on the external validity of results [6]. Therefore, we conclude that it is likely the limited level of experience of the participants in the experiment did not distort the study results.

The instrumentation in the experiment might become unrealistic or old-fashioned in future. The quantity and quality of recovered links in the experiment were representative for state-of-the-art manual, semi-automatic and automatic traceability tools in general. However, a threat to validity remains that the measured cost-effects of the automation alternatives of traceability link recovery cannot be 1:1 translated to all future tools.

Construct validity. The fact that only one software system (the Apache Wink) is used in the experiment, introduces the risk that the cause construct is under-represented. In this experiment, we consider that the used system is representative for large and medium-size object-oriented systems. The threat, however, cannot totally be ignored.

Conclusion validity. A threat to validity might result from the interpretation of the architecture-level software understanding activities because answers of these activities consists of a list of system elements. We mitigated this risk by calculating the standard information retrieval metrics for retrieved elements from all understanding activities. We argue that information retrieval measures allow analysts to objectively evaluate the correctness of particular activities rather than intuitive or ad-hoc human measures. This potential threat is mitigated to large degree.

Finally, the violation of assumptions made by statistical tests could distort the results of the experiment. Due to the violation of the homogeneity assumption, the pairwise Wilcoxon Rank-Sum test after a Kruskal-Wallis test is used to test the significance of the found results. Note that the

results of these tests were interpreted as statistically significant at $\alpha = 0.05$ (i.e., the level of confidence is 95%). Thus, this factor is not seen as a threat to validity.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we describe the results of a controlled experiment that was conducted to empirically investigate in how far the use of different automation alternatives for traceability link recovery supports architecture-level understanding of the software system. Three aspects were specifically taken into consideration: the quantity of correctly and incorrectly retrieved elements, and their overall quality. The results from the semi-automatic group in the experiment reveal a focus on a traceability-based assessment process, which was mainly driven by exploring the imports and source code packages of the main class(es). The answers of the automatic group show a focus on finding out the correct traceability links. It is hard for them to further explore the remaining set of relevant links. This might stem from the fact that automated approaches produce inaccurate and incomplete traceability links, and tend to result in a low precision and proliferation of traceability links that are difficult to manage and understand. The participants of the manual group also used traceability links to identify the high-level features in the code classes and vice versa.

The evaluation of the experiment showed that the highest quantity and quality of elements can be achieved using both manual and semi-automatic traceability methods. Based on our experiences and observations from various open-source software systems, we have identified that the problems with manual and automatic traceability can be mitigated and resolved with semi-automatic traceability approaches. For example, the output produced by the semi-automatic methods can be observed to discard the incorrect links and to cover the remaining set of missing links. If cost and effects are considered upfront, the semi-automatic traceability appears to be more cost-effective solution than the other two alternatives of traceability link recovery.

As it is usual for empirical studies, replications in different contexts, with different objects and participants, are good ways to corroborate our findings. Replicating the experiment with different objects (software systems) of varying sizes and complexity, and different participants (seasoned software architects and masters students) are part of our future work agenda. Another direction for future work is to investigate the automation alternatives of traceability links at fine-grained level of granularity.

7. ACKNOWLEDGEMENTS

This work is supported by the Austrian Science Fund (FWF), under project P24345-N23. We also thank to all the participants for taking part in the experiment.

8. REFERENCES

[1] D. Cuddeback, A. Dekhtyar, and J. Hayes. Automated requirements traceability: The study of human analysts. In *Proceedings of the 18th International Requirements Engineering Conference*, RE '10, pages 231–240. IEEE.

[2] G. A. A. Cysneiros, F. Andrea, and Z. G. Spanoudakis. Traceability approach for i* and uml models. In *Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'03)*, 2003.

[3] A. Dekhtyar, O. Dekhtyar, J. Holden, J. Hayes, D. Cuddeback, and W.-K. Kong. On human analyst performance in assisted requirements tracing: Statistical analysis. In *Proceedings of the 19th International Requirements Engineering Conference*, RE '11, pages 111–120. IEEE.

[4] A. Egyed, S. Biffl, M. Heindl, and P. Grünbacher. A value-based approach for understanding cost-benefit trade-offs during automated software traceability. In *Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE 2005, pages 2–7. ACM.

[5] M. A. Javed and U. Zdun. On the effects of traceability links in differently sized software systems. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE 2015. ACM.

[6] M. A. Javed and U. Zdun. The supportive effect of traceability links in architecture-level software understanding: Two controlled experiments. In *Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture*, WICSA 2014, pages 215–224. IEEE.

[7] M. A. Javed and U. Zdun. The supportive effect of traceability links in change impact analysis for evolving architectures – two controlled experiments. In *14th International Conference on Software Reuse*, ICSR 2015. Springer link.

[8] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, Aug. 2002.

[9] S. Klock, M. Gethers, B. Dit, and D. Poshyvanyk. Traceclipse: An eclipse plug-in for traceability link recovery and management. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '11, pages 24–30. ACM.

[10] W. H. Kruskal and W. A. Wallis. Use of Ranks in One-Criterion Variance Analysis. volume 47, pages 583–621. American Statistical Association, 1952.

[11] H. Mann and D. Whitney. On a test of whether one of two random variables is stochastically larger than the other. volume 18, pages 50–60. Institute of Mathematical Statistics, 1947.

[12] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. On the equivalence of information retrieval methods for automated traceability link recovery. In *Proceedings of the 18th International Conference on Program Comprehension*, ICPC '10, pages 68–71. IEEE.

[13] C. Wohlin. *Experimentation in Software Engineering: An Introduction: An Introduction.* The Kluwer International Series in Software Engineering. Kluwer Academic, 2000.

[14] X. Zou, R. Settimi, and J. Cleland-Huang. Improving automated requirements trace retrieval: A study of term-based enhancement methods. *Empirical Softw. Engg.*, 15(2):119–146, Apr. 2010.