The Supportive Effect of Traceability Links in Change Impact Analysis for Evolving Architectures – Two Controlled Experiments

Muhammad Atif Javed and Uwe Zdun

Software Architecture Research Group
University of Vienna, Austria.
{muhammad.atif.javed,uwe.zdun}@univie.ac.at

Abstract. The documentation of software architecture relations as a kind of traceability information is considered important to help people understand the consequences or ripple-effects of architecture evolution. Traceability information provides a basis for analysing and evaluating software evolution, and consequently, it can be used for tasks like reuse evaluation and improvement throughout the evolution of software. To date, however, none of the published empirical studies on software architecture traceability have examined the validity of these propositions. In this paper, we hypothesize that impact analysis of changes in software architecture can be more efficient when supported by traceability links. To test this hypothesis, we designed two controlled experiments that were conducted to investigate the influence of traceability links on the quantity and quality of retrieved assets during architecture evolution analysis. The results provide statistical evidence that a focus on architecture traceability significantly reduces the quantity of missing and incorrect assets, and increases the overall quality of architecture impact analysis for evolution.

Keywords: software architecture traceability, architecture evolution, change impact analysis, empirical software engineering, controlled experiment

1 Introduction

During the last four decades, many investigations on software change impact analysis techniques and its applications have been performed [17, 19]. Change impact analysis is about determining the consequences or ripple-effects of proposed changes in the software system. To support software evolution, architectural change impact analysis is considered of great importance as understanding the architecture and its changes is a foundation of software evolution analysis at the architectural level [12]. The architectural level is well suited for software evolution analysis, as the software architecture allows (early) reasoning on the quality attributes of the system [4] and the software architecture not only describes the high-level structure and behaviour of the system, but also incorporates principles and decisions that determine the system's development and its evolution [2].

Software change impact analysis techniques and their applications are based on either traceability information or dependence relationships. These techniques do not only provide a basis for analysing and evaluating software evolution, but can also be used for tasks such as reuse evaluation and improvement. For example, the identified impacts of specific software assets from architecture evolution analysis can be used as a basis to examine the level of reuse of those assets throughout a number of evolution steps. It is also pointed out by Selby [13] that, in general, software assets reused without or with limited revisions have fewer faults than software assets reused with major revisions. Hence, understanding change impact provides the means to control reuse and evolution of software assets.

Traceability links between the software architecture and other software assets, such as the source code or the requirements, are considered important to determine the potential evolution impacts at the architectural level [8]. However, none of the published empirical studies on software architecture traceability provide quantitative evidence of the added value of traceability links in the evolution of software architectures. To date, two empirical studies on architecture traceability have been published [7, 14]. These studies mainly concern the understanding of architecture designs. The lack of published empirical data on the benefits of architecture traceability is one of the reasons that prevents the wide adoption of traceability approaches in industrial settings [7, 14]. It is crucial to conduct more empirical studies on the usefulness of architecture traceability to find out whether the use of architecture traceability can significantly support the development activities in order to justify its costs.

The goal of this paper is to empirically validate whether change impact analysis is more efficient regarding the software architecture evolution activities, if the impact analysis is supported by traceability links. In particular, we intend to answer the following research question: Are the quality and quantity of retrieved assets during software architecture change impact analysis higher for change impact analysis that is supported by traceability links than for change impact analysis without traceability links? Note that the assets to be retrieved during our experiments are source code classes and components in a component model that are affected by a change.

To answer the research question, we conducted two controlled experiments at the University of Vienna, Austria, in May 2014. The first experiment was carried out with 51 students, whereas the other 56 students participated in the second experiment. They were asked to perform seven impact understanding activities that concern the evolution at the architectural level. In both experiments, half of the students were asked to perform the impact analysis of changes in software architecture by using the information from the architectural documentation and the source code of the system, while the other half performed the same tasks with the same provided information and additionally received traceability links between the architectural models and the source code. The former group is referred to as the control group, the latter as the experiment group. The data from the experiments was analysed, and the quantity of missing and erroneous retrieved assets during the architecture evolution analysis and their overall quality were compared. The results of the experiments provide strong evidence for the benefits of using traceability links concerning the quantity and quality of the assets retrieved during change impact analysis activities for evolving software architectures.

The rest of this paper is organized as follows: Section 2 describes the related work. Section 3 discusses the design of the controlled experiments including the introduction of variables and hypotheses, while the subsequent Section 4 explains the details concerning the execution of the experiments. Section 5 presents the hypotheses tested and the analysis of the results of the study. Section 6 contains the interpretation of the findings and a discussion of threats to validity. Section 7 concludes the study and discusses future work.

2 Related Work

As mentioned in the introduction, there exist only two earlier studies, one performed by our research group [7] and one by Shahin et al. [14], that provide quantitative evidence of the added value of traceability links in understanding of architecture designs.

In our own previous study [7] we conducted a controlled experiment and its replication to evaluate the support provided by traceability links between architectural models and the source code. The experiments were conducted with 108 participants. The participants were asked to answer twelve typical questions aimed at gaining an architecture-level understanding of a representative subject system, with and without traceability information. 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 are observed.

The work by Shahin et al. [15] analyse the support provided by Compendium tool [14], a tool to visualize architectural design decisions and their rationale, as a kind of traceability information. The experiment was carried out with 10 participants. The participants were asked to understand the existing design and to make the new design according the new requirement, with and without Compendium tool. The results show that Compendium significantly improves the correctness of understanding architecture design in architecting process, and does not increase the total time for reading software architecture documentations and performing design task.

The contribution of this study is novel for two main reasons. First, there exist no published evidence related to the added value of traceability links in software architecture evolution. Second, most of the earlier works are based on some specific traceability tools, which do not enable a distinction between tool support and the usefulness of traceability links. In our experiments, for practical reasons and to study the foundational concepts rather than a specific tool, the participants were provided with hyperlink-based access of traceability links and the source code, to investigate the support provided by traceability links between architectural models and the source code in evolution of software system architectures, rather than the support provided by a specific tool.

3 Design of the Experiment

For the study design, the guidelines for experiments' conduct by Kitchenham et al. [10] and Wohlin et al. [18], and reporting by Jedlitschka and Pfahl [9] were used. Kitchenham et al. present preliminary guidelines for experimentation in software engineering and give some instructions regarding the context, design, data collection, analysis, presentation, and interpretation of empirical studies without going into detail. Wohlin et al. present the experiment phases in more detail, and also discuss statistical tests and their suitability for different kinds of studies. The former guidelines were primarily used in the planning phase of our experiments, while the latter was used as a reference for the analysis and interpretation of the results. Jedlitschka's and Pfahl's guidelines for reporting controlled experiments are used to describe the experiments in this paper. Please note that the following subsections of the reporting template were omitted, because they were either not applicable, or their content was already mentioned in other sections: Relation to existing evidence is presented in Section 2; inferences and lessons learned are discussed in Section 6; interpretation and general limitations of the study are described in Section 6.2.

3.1 Goal, hypotheses, parameters, and variables

The goal of the experiments is to empirically investigate, if change impact analysis that is based on traceability links significantly reduces the quantity of missing and incorrect retrieved assets, and increases their overall quality during the architecture evolution analysis. The experiments goal led to the following null hypotheses and corresponding alternative hypotheses:

H_{01}: The use of traceability links *does not significantly increase* the quantity of correctly retrieved assets during architecture evolution analysis.
H_1: The use of traceability links *significantly increases* the quantity of correctly retrieved assets during architecture evolution analysis.
H_{02}: The use of traceability links *does not significantly reduce* the quantity of incorrectly retrieved assets during architecture evolution analysis.
H_2: The use of traceability links *significantly reduces* the quantity of incorrectly retrieved assets during architecture evolution analysis.
H_{03}: The use of traceability links *does not significantly increase* the overall quality of retrieved assets during architecture evolution analysis.
H_3: The use of traceability links *significantly increases* the overall quality of retrieved assets during architecture evolution analysis.

Description	Scale Type	Unit	Range
Quantity of correctly retrieved assets	Interval	Points	[0 - 1]
Quantity of incorrectly retrieved assets	Interval	Points	[0 - 1]
Overall quality of the retrieved assets	Interval	Points	[0 - 1]

Table 1: Dependent Variables

Dependent variables Three dependent variables were observed during the experiments, as shown in Table 1: the quantity of correctly and incorrectly retrieved assets, and their overall quality, in the architecture evolution analysis. They were accessed by using the standard information retrieval metrics, in particular, recall, precision, and f-measure, respectively [1, 6]. Because impact analysis of changes in software architecture consists of a list of system assets, two aspects were specifically taken into consideration to measure the recall and precision of the retrieved assets:

- The set of *correct assets* expected in the solution to *activity a* (C_a).
- The set of *assets retrieved* in the solution to *activity a* by *participant p* ($R_{p,a}$).

$$Recall_{p,a} = \frac{\mid C_a \cap R_{p,a} \mid}{C_a} \qquad Precision_{p,a} = \frac{\mid C_a \cap R_{p,a} \mid}{R_{p,a}}$$

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 architecture change impact analysis activities from the experiments' participants.

Description	Scale Type	Unit	Range/Possible Values
Time	Ordinal	Minutes	90 minutes (Max)
Group Affiliation	Nominal	N/A	Control group, Experiment group
Programming experience	Ordinal	Years	4 classes: 0-1, 1–3, 3–7, >8
Architecture experience	Ordinal	Years	4 classes: 0-1, 1–3, 3–7, >8
Affiliation	Nominal	N/A	Academia, Industry, Other

Table 2: Independent Variables

Independent variables Five independent variables were observed during the experiments, as shown in Table 2. They relate to the personal information (programming experience, architecture experience, affiliation), group affiliation (control group or experiment group) and time spent in the experiments. These variables could have an influence on the dependent variables, which is eliminated by balancing the characteristics between the control groups and the experiment groups in the same way, in particular, through random assignment to the two groups in both experiments.

3.2 Experiment Design

To test the hypotheses, we conducted two controlled experiments [3] at the University of Vienna, Austria, in May 2014. The experiments were conducted as practical sessions on architecture evolution analysis.

Participants The participants in the experiments were 107 individual students of the software architecture course held at University of Vienna. The first experiment was conducted with 51 students, while the other 56 students had participated in the second experiment.

Objects The basis for the architecture impact recovery was UltraESB[1] Version 2.3.0 and PetalsESB[2] Version 4.2.0. Both systems belong to the enterprise service bus (ESB) domain, which provides an connectivity infrastructure to integrate the services within a service-oriented architecture.

Blocking To be able to explicitly analyse the influence of traceability links in change impact analysis of software architecture evolution, the participants in both experiments were randomly assigned to the two balanced groups. For each experiment, one group

[1] `http://adroitlogic.org/products/ultraesb.html`

[2] `http://petals.ow2.org`

of participants was asked to determine the impact of architecture evolution activities by using the information from the architectural documentation and the source code of the system, whereas the other group performed the same tasks, but additionally received the traceability links between architectural models and the source code. The first group is referred to as control group, the latter as experiment group.

Instrumentation To obtain the necessary data related to the influence of traceability links in architecture evolution analysis, the instruments discussed in the following paragraphs were used to carry out the experiments.

Three pages of architectural documentation about the used objects: The participants in the first experiment were provided with the documentation for UltraESB, while the participants of the second experiment received the documentation for PetalsESB. The documentation describes the conceptual architecture 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.

Web-based access for the source code: The participants in the first and second experiment were provided with the web-based access of syntax-highlighted source code for the UltraESB and PetalsESB, respectively. The cover page alphabetically lists the source code package names and their enclosed code classes, and provides a hyperlink-based support to 'jump' to specific assets (code classes or packages) located in the Git repository[3]. The participants in the experiment groups were also provided with the similar support for traceability links, represented as lists: Each entry in a list contains information about architectural components and their realized code classes, which represent individual traceability link.

A questionnaire to be filled-in by the participants during the experiments: 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 seven architecture impact analysis activities, as shown in Table 3. In the context of these activities, two important criteria are applied: (i) the activities should be representative for key architecture impact analysis and evolution contexts for both UltraESB and PetalsESB, and (ii) they should be imaginatively constructed to measure the deeper impact understanding from participant groups. Note that the same impact evaluation activities listed in Table 3 were used for both UltraESB and PetalsESB, which was possible as different ESBs share many similar architectural concerns. The results expected from participants for each activity were sets of retrieved asset names (i.e., names of source code classes and components from the provided component models).

Blinding To eliminate subjective bias on the part of both experiments' participants and the experimenters, double blinding was applied in the experiments. Although, participants perceived that there are two different groups for each experiment, they were not aware about the purpose of group division and their group affiliation.

[3] `http://git-scm.com`

ID	Description
A1	Investigate the impact of extensions in the transport senders and listeners
A2	Investigate the consequences of extensions in the traffic monitoring
A3	Determine the ripple-effects of changes in the ESB configuration
A4	Investigate the impact of changes in the message interception
A5	Evaluate the effects of high availability and capacity of ESB server
A6	Investigate the consequences of new message endpoints
A7	Determine the impact of new deployment aspect implementation

Table 3: Impact evaluation activities at the architectural-level (used for both UltraESB and PetalsESB)

The results of the experiments were handed over to two independent researchers who did not know the real identity of the participants. This was done to prevent the experiments from being biased. To be able to compute the results of the change impact analysis of retrieved assets, the researchers were asked to compute the information retrieval statistics by matching the participants' answers with the original solution model. This allows us to objectively evaluate the quantity and quality of the retrieved assets rather than by intuitive or ad-hoc human measures.

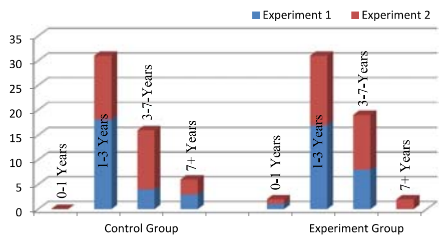
Data collection procedure After introduction and grouping, the participants received the instruments, mentioned in Section 3.2. The provided instruments had to be used to perform the impact analysis of architecture evolution activities. The participants were distributed over separate rooms according to their group membership. At least one experimenter was present in each room to answer the questions related to the instructions and to restrict the participants from consulting others and using forbidden material. The participants were given 90 minutes to determine the ripple effects of architecture evolution activities. After completion of the session, the filled-in questionnaires were collected by the experimenters and finally a discussion in the wrap-up phase was arranged to gather further information from the participant groups. All the participants were present during the discussion.

4 Execution

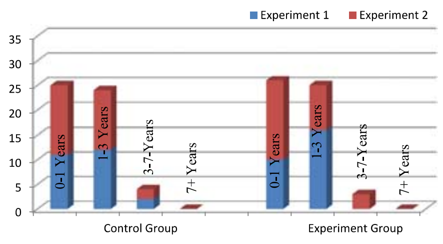
4.1 Sample and Preparation

As described in Section 3.2, the experiments were conducted in two practical sessions on architecture evolution analysis at the University of Vienna, Austria. The first experiment took place with 51 students of the software architecture course; the second experiment was conducted with another 56 students of the same course.

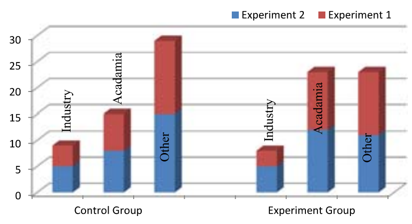
Figure 1 shows the distribution of the participants based on their previous experience and affiliation, as assigned to the control group and the experiment group. The data presented in the figures was accumulated from all the participants in the two experiments, but also shows the separate data of the experiments. 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



(a) Programming Experience



(b) Architecture Experience



(c) Affiliation

Fig. 1: Distribution of participants

previous experiences in the control groups is slightly better both regarding programming and architecture. In the experiment groups slightly more people with an academic affiliation and slightly less with an industry affiliation are present. However, overall the experiences and affiliations are rather well balanced in the two experiments.

4.2 Data collection performed

The data collection procedure was performed as planned in the study design. There were no participants who dropped out and no deviations from the study design occurred.

4.3 Validity procedure

The experiments were conducted in a controlled environment. The participants in both experiments were assigned to different rooms according to their group membership (control group or experiment group). The participants in each rooms were supervised by at least one experimenter during the whole duration, enabling them to ask clarification questions and restrict them from talking to each other or using forbidden material. All the participants had to return the questionnaire before leaving the room. The filled-in questionnaire were collected from the remaining participants after completion of experiments' sessions. No unexpected situation occurred during the experiments.

5 Analysis

5.1 Descriptive statistics

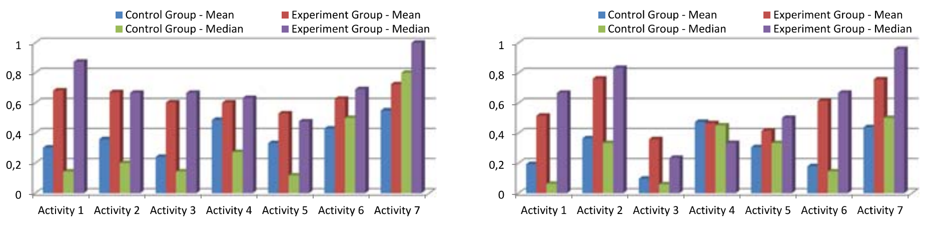
The descriptive statistics shows the results of the experiments as a first step in the analysis. The first two subsections concern the quantity of correctly and incorrectly retrieved assets respectively. The last subsection presents an analysis of the overall quality of retrieved assets during architecture evolution analysis.

Quantity of correctly retrieved assets The descriptive statistics for the quantity of correctly retrieved assets for the control groups and the experiment groups from the two experiments are shown in Table 4 and Figure 2. The data in the table is based on the sum of the recall of the experiments' activities for each participant, while the figure concerns the recall for each experiment activity.

Execution	Group Affiliation	Mean	Median	Std. Dev.
Experiment 1	Control Group	2.701009 (0.3858584 %)	2.565584 (0.3665121 %)	1.808094 (0.2582992 %)
	Experiment Group	4.439981 (0.634283 %)	4.868956 (0.6955651 %)	1.708463 (0.2440661 %)
Experiment 2	Control Group	2.04751 (0.2925014 %)	1.908818 (0.2726883 %)	1.023914 (0.1462735 %)
	Experiment Group	4.114883 (0.5878404 %)	4.491484 (0.6416406 %)	1.421371 (0.203053 %)

Table 4: Descriptive analysis of the quantity of correct retrieved assets

As we see from Table 4, the total quantity of correctly retrieved assets is higher in the experiment groups than in the control groups. The results in Figure 2 show that the participants of the experiment group belonging to the first experiment have a higher number of correctly retrieved assets for all impact analysis activities than the control group. However, the participants of the control group of the second experiment have outperformed the participants of the experiment group in Activity 4.



(a) Experiment 1 (b) Experiment 2

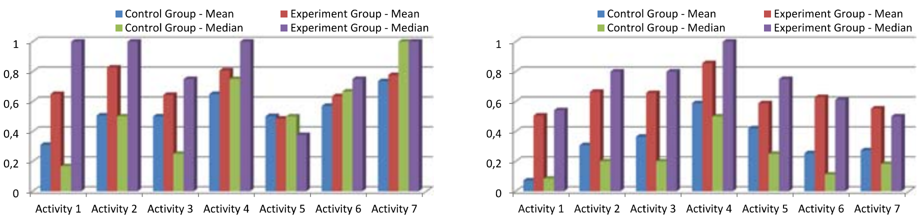
Fig. 2: Quantity of correctly retrieved assets for each experiment activity

Quantity of incorrectly retrieved assets Table 5 and Figure 3 show the comparisons for the quantity of retrieved assets that are actually correct for the control groups and the experiment groups in the two experiments. The data in the table is based on the sum of the precision values of the experiments' activities for each participant. In total, the quantity of retrieved assets that are actually correct is higher in the experiment groups than the control groups. As a consequence, this means that the quantity of incorrectly retrieved assets is lower in the experiment groups compared to control groups.

Execution	Group Affiliation	Mean	Median	Std. Dev.
Experiment 1	Control Group	3.774333 (0.5391905 %)	3.208333 (0.4583333 %)	1.779565 (0.2542236 %)
	Experiment Group	4.826603 (0.6895147 %)	4.6875 (0.6696429 %)	1.865917 (0.2665595 %)
Experiment 2	Control Group	2.278481 (0.3254973 %)	2.242929 (0.3204185 %)	1.191005 (0.1701435 %)
	Experiment Group	4.454726 (0.6363894 %)	4.523485 (0.6462121 %)	1.593693 (0.2276704 %)

Table 5: Descriptive analysis of the quantity of actually correctly retrieved assets

The results in the Figure 3 concern the precision for each experiment activity, in which the participants of the control group only outperformed the participants of the experiment group in Activity 5 of the first experiment.



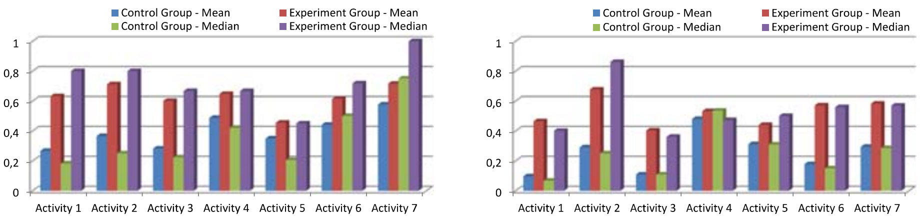
(a) Experiment 1 (b) Experiment 2

Fig. 3: Quantity of actually correctly retrieved assets for each experiment activity

Overall quality of retrieved assets The descriptive statistics for the overall quality of retrieved assets for the control groups and the experiment groups from the two experiments is shown in Table 6 and Figure 4. The results in the table are based on the sum of the overall quality of retrieved assets (i.e., the f-measure) of the experiments' activities for each participant, while the figure shows the f-measure results for each experiment activity. The data in the table and figure show that the average quality of retrieved assets in the experiment groups seems to be higher than the average quality of retrieved assets in the control groups.

Execution	Group Affiliation	Mean	Median	Std. Dev.
Experiment 1	Control Group	2.767399 (0.3953427 %)	2.516986 (0.3595694 %)	1.624837 (0.2321195 %)
	Experiment Group	4.377607 (0.6253725 %)	4.275092 (0.6107274 %)	1.722879 (0.2461256 %)
Experiment 2	Control Group	1.755936 (0.2508479 %)	1.769355 (0.252765 %)	0.7923499 (0.1131928 %)
	Experiment Group	3.66901 (0.5241442 %)	3.608125 (0.5154464 %)	1.617311 (0.2310445 %)

Table 6: Descriptive analysis for the overall quality of retrieved assets



(a) Experiment 1 (b) Experiment 2

Fig. 4: Overall quality of retrieved assets for each experiment activity

Dataset reduction Outliers in the dataset, i.e., data points that are either much lower or much higher than other data points, are potential candidates for dataset reduction. Thirteen of the participants from the two experiments did not perform all the activities. This results in nineteen missing data points in the experiments. As it seems that these participants have spend sufficiently longer time in exploring the source code, we have not excluded these data points from the study.

To find potential outliers, we also calculated the quantity and quality of the architecture evolution activities for each participant. Note that four of the participants from the experiment groups reached a considerable lower quantity and quality of architecture evolution analysis activities than the other members of these group. A closer analysis showed that they could not properly make use of traceability links to perform the architecture impact understanding activities. However, their results were not excluded as outliers, because the difference to the other participants is not strong enough. Excluding these data points would have introduced a potential vulnerability of the study results.

5.2 Analysis of the opinion of participants

This subsection summarizes the results of the wrap-up discussion phase which was arranged after each experiment session to gather further information from the participant groups.

The participants in the experiment groups from the two experiments and the control group of the first experiment have acknowledged that they had enough time to perform the architecture evolution analysis activities. However, the participants in the control group of the second experiment showed concerns related to the provided time for performing the activities. This is probably because the second experiment was conducted with a rather large system (PetalsESB) compared to first experiment (UltraESB). The same happened also for the experience and difficulties of the participants: The participants of the control groups experienced more difficulties in performing the activities than the participants of the experiment groups, in addition, the participants with '0-1 years' of experience encountered more difficulties than the participants with '1-8+ years' of experience.

The participants were also asked about their familiarity with the application domain. The answers imply that enterprise service bus, which is the application domain of the UltraESB and PetalsESB, is well-known to the participants from previous lectures of the software architecture course.

The next two questions concerned the usage and helpfulness of traceability links for architecture evolution analysis. First, the participants were asked whether traceability links are useful in impact analysis of changes in software architecture. The answers reflect that the participants had knowledge about traceability links. The members of both groups generally consider traceability links as useful in architecture-level impact understanding of the software system. In the next question the participants were asked whether they used traceability links before. The answers show that only a very few participants have previously used traceability links for understanding of software assets outside of the lecture in which the experiments took place.

Finally, the participants were asked to briefly describe how the architecture evolution analysis was performed. This was primarily done to confirm that the experiment groups used the traceability links and to find out if the control groups used any other systematic way to perform architecture evolution analysis. The answers of the control groups reveal a focus on an intuitive approach, which was mainly driven by personal experience or judgements. The respondents stated that they performed the activities by reading the textual description in the architecture document and intuitively exploring the code classes. They acknowledged that it is hard to find the correct links between architecture and implementation artefacts. This might stem from the fact that software architecture is not explicitly represented in the code classes, e.g. as packages and classes or similar code-level abstractions. The answers of the experiment groups show a focus on the traceability links. The respondents of experiment groups stated that they used traceability links to identify the ripple-effects of architecture artefacts in the code classes and vice versa. They confirmed that they primarily used this additional knowledge for performing architecture impact analysis for evolution.

5.3 Hypothesis testing and results

Quantity of correctly retrieved assets To be able to test the first null hypothesis H_{o1}, the influence of traceability links on the quantity of correctly retrieved assets is measured. In the analysis of the experiments, the Shapiro-Wilk normality test [16] and Wilcoxon Rank-Sum test [11] are used. First, the Shapiro-Wilk normality test is used to find out whether equal variances of the level of correctness can be assumed. Second, as a consequence of non-normal distributions, the corresponding non-parametric statistical test, Wilcoxon Rank-Sum test, is used to test the significance of the found results. Note that the results of tests were interpreted as statistically significant at $\alpha = 0.05$ (i.e., the level of confidence is 95%).

Execution	**Factor**	**Wilcoxon Rank-Sum Test**
Experiment1	Control Group vs. Experiment Group	W = 492, **p-value = 0.001332**
Experiment 2	Control Group vs. Experiment Group	W = 686.5, **p-value = 0.000001451**

Table 7: Wilcoxon-test for quantity of correct retrieved assets

Table 7 shows the results of the Wilcoxon rank-sum test for the control groups and the experiment groups. The table shows that both experiments (Experiment 1 and Experiment 2) provide strong evidence that H_{o1} can be rejected. This means that in our two experiments the use of traceability links significantly improved the quantity of correctly retrieved assets during the architecture evolution analysis.

Quantity of incorrectly retrieved assets Hypothesis H_{o2} was also evaluated with a Wilcoxon rank-sum test. The results are shown in Table 8. The table shows that the both experiments (Experiment 1 and Experiment 2) provide strong evidence that H_{o2} can be rejected. This means that in our two experiments the use of traceability links significantly reduces the quantity of incorrectly retrieved assets in the architecture impact analysis for evolution. Note that there is a noticeable difference in the p-values between the two experiments. The main reason behind this difference probably is the varying sizes and complexity of the objects (software systems) in the first and second experiment.

Execution	**Factor**	**Wilcoxon Rank-Sum Test**
Experiment 1	Control Group vs. Experiment Group	W = 437.5, **p-value = 0.03446**
Experiment 2	Control Group vs. Experiment Group	W = 661, **p-value = 0.00001083**

Table 8: Wilcoxon-test for quantity of incorrect retrieved assets

Overall quality of retrieved assets The Wilcoxon rank-sum test is also used to evaluate the Hypothesis H_{o3}. The results are shown in Table 9. The table shows that the both experiments provide strong evidence that H_{o3} can be rejected. This means that in our two experiments the use of traceability links significantly improved the overall quality of the retrieved assets for architecture evolution analysis.

Execution	Factor	Wilcoxon Rank-Sum Test
Execution 1	Control Group vs. Experiment Group	W = 497, **p-value = 0.0009243**
Execution 2	Control Group vs. Experiment Group	W = 659, **p-value = 0.000004097**

Table 9: Wilcoxon-test for overall quality of retrieved assets

6 Interpretation

6.1 Evaluation of results and implications

As discussed in the previous section, all three null hypotheses can be rejected and hence we can deduce the following implications:

- As H_{o1} can be rejected, according to our experiments, there is evidence that the quantity of correctly retrieved assets during the architecture evolution analysis is higher if traceability links are used.
- As H_{o2} can be rejected, according to our experiments, there is evidence that the quantity of incorrectly retrieved assets during the architecture evolution analysis is lower if traceability links are used.
- As H_{o3} can be rejected, according to our experiments, there is evidence that the quality of retrieved assets for architecture evolution analysis is higher if traceability links are used.

6.2 Threads to validity and limitations of the study

Multiple levels of validity threats have to be considered in the experiments. We have considered the classification scheme for validity in experiments by Cook and Campbell [5]. The internal validity concerns the cause effect inferences between the treatment and the dependent variables measured in the experiments. External validity refers to the generalizability of the results for a larger population. Construct validity is about the suitability of the study design for the theory behind the experiments. Finally, conclusion validity focuses on the relationship between treatment and outcome and on the ability to draw conclusions from this relationship. All validity threats in the experiments are categorized based on this classification.

Internal validity

- The architecture evolution analysis activities could have been biased towards the experiment groups. The threat, however, is mitigated by considering many characteristics of software architecture evolution. As a result, the change impact analysis activities concerned both architecture recovery and evolution contexts. Therefore, we do not consider it a highly relevant threat to validity.
- The analysts in the experiments could have graded the retrieved assets incorrectly. We tried to mitigate this risk by providing the original solution model to the analysts. The analysts were asked to apply the solution model to the recovered participants' solutions. The solution model clearly states the correct assets for each architecture evolution analysis activity. Furthermore, the results have also been verified by the authors of this paper.

- Finally, the analysts could have been biased towards the experiment groups. We tried to exclude this threat to validity by not revealing the identity of the participants or in which of the two 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 experiments were conducted with rather inexperienced participants, the students of a software architecture 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 [7]. Therefore, we conclude that it is likely the limited level of experience of the participants in the two experiments does not distort the study results.
- The instrumentation in the experiments might have been unrealistic or old-fashioned. In this case, the architecture evolution analysis was based on the hyperlinks. In practice, different tools would be used to support evolution analysis. These tools are primarily used to formulate and maintain the traceability or dependence relationships between the related software assets. In these experiments, for practical reasons and to study the foundational concepts rather than a specific tool, the source code of the software systems and traceability links were readily provided in a web-based format. We assume that the measured effect of the experiment groups during traceability recovery is independent of the way in which a tool would visualize the traceability links, but a threat to validity remains that our results cannot be 1:1 translated to all existing tools and visualizations.

Construct validity

- The use of one object in the experiment introduces the risk that the cause construct is under-represented. In this case, the experiments were conducted with different objects, in particular, the UltraESB and PetalsESB, although the objects belong to the same domain but represent software systems, of significantly different size (in terms of number of source code classes). The threat, however, cannot totally be ignored.
- Another potential threat to validity is the number of measures used to evaluate the quantity and quality of retrieved assets. In our case we only used standard information retrieval metrics, in particular, recall, precision, and f-measure, to measure the quantity of correctly and incorrectly retrieved assets, and their overall quality, respectively. This does not allow for cross-checking the results with different measures.

Conclusion validity

- A threat to validity might result from the interpretation of the architecture evolution analysis activities because impact of these activities consists of a list of system assets (e.g., architectural components, source code classes). We mitigated this risk by calculating the standard information retrieval metrics for retrieved assets from all architecture evolution analysis activities. We argue that information retrieval measures allow analysts to objectively evaluate the correctness of architecture evolution analysis activities rather than intuitive or ad-hoc human measures. We conclude that this potential threat is mitigated to large degree.

- Finally, the violation of assumptions made by statistical tests could distort the results of the experiments. In the analysis of the experiments, the Shapiro-Wilk normality test and Wilcoxon Rank-Sum test are used. First, Shapiro-Wilk normality test is used to find out whether equal variances of the level of correctness can be assumed. Second, as a consequence of non-normal distributions, the corresponding non-parametric statistical test, the Wilcoxon Rank-Sum test, is used to test the significance of the found results. Note that the results of the 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.

7 Conclusions and future work

In this paper, we describe the results of two controlled experiments that were conducted to find out if traceability links are beneficial for change impact analysis of evolving architectures. Three aspects were specifically taken into consideration: the quantity of correctly and incorrectly retrieved assets, and their overall quality. The evaluation of the experiments shows that using traceability links leads to significantly lower quantity of missing and incorrect assets, and overall, a higher quality of architecture evolution analysis. Because the calculation procedure for architecture-centric reuse evaluation, with the focus on traceability links, is carried out in a similar manner to the calculation of the architecture evolution analysis, it is likely that the results can be generally applicable for architecture-centric reuse of the software systems making use of traceability links.

As it is usual for empirical studies, replications in different contexts, with different objects and participants, are good ways to corroborate our findings. Comparing the results of the different objects (software systems) in terms of their sizes and complexity is part of our future work agenda. Another direction for future work is to replicate the experiment with our evoluation and reusability evaluation tool that is currently under development.

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 experiments.

Bibliography

[1] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999.

[2] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet. Architecture-level modifiability analysis (alma). *J. Syst. Softw.*, 69(1-2):129–147, Jan. 2004.

[3] B. Boehm, V. Basili, H. Rombach, and M. Zelkowitz. *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili.* Springer, 2005.

[4] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures: methods and case studies.* SEI series in software engineering. Addison-Wesley, 2002.

[5] T. Cook and D. Campbell. *Quasi-experimentation: design & analysis issues for field settings.* Houghton Mifflin, 1979.

[6] D. Harman. Ranking algorithms. In W. B. Frakes and R. Baeza-Yates, editors, *Information retrieval*, pages 363–392. Prentice-Hall, 1992.

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

[8] M. A. Javed and U. Zdun. A systematic literature review of traceability approaches between software architecture and source code. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 16:1–16:10, New York, NY, USA, 2014. ACM.

[9] A. Jedlitschka and D. Pfahl. Reporting guidelines for controlled experiments in software engineering. In *2005 International Symposium on Empirical Software Engineering (ISESE 2005)*, pages 95–104, 2005.

[10] 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. *Software Engineering, IEEE Transactions on*, 28(8):721–734, Aug. 2002.

[11] H. Mann and D. Whitney. *On a Test of Whether One of Two Random Variables is Stochastically Larger Than the Other.* Institute of Mathematical Statistics, 1947.

[12] T. Mens, J. Magee, and B. Rumpe. Evolving software architecture descriptions of critical systems. *IEEE Computer*, 43(5):42–48, 2010.

[13] R. Selby. Enabling reuse-based software development of large-scale systems. *Software Engineering, IEEE Transactions on*, 31(6):495–510, June 2005.

[14] M. Shahin, P. Liang, and M. R. Khayyambashi. Rationale visualization of software architectural design decision using compendium. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2367–2368, New York, NY, USA, 2010. ACM.

[15] M. Shahin, P. Liang, and Z. Li. Architectural design decision visualization for architecture design: preliminary results of a controlled experiment. In *Proceedings of the 5th European Conference on Software Architecture: Companion Volume*, ECSA '11, pages 2:1–2:8, New York, NY, USA, 2011. ACM.

[16] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):pp. 591–611, 1965.

[17] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Syst. J.*, 13(2):115–139, June 1974.

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

[19] S. Yau, J. Collofello, and T. MacGregor. Ripple effect analysis of software maintenance. In *Computer Software and Applications Conference, 1978. COMPSAC '78. The IEEE Computer Society's Second International*, pages 60–65, 1978.