# The Supportive Effect of Traceability Links in Architecture-Level Software Understanding: 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 advocates of architecture traceability approaches regularly cite advantages like easier understanding of architectural designs and support for software quality control and maintenance. However, the lack of published empirical data on the usefulness of architecture traceability is one of the reasons that prevents the wide adoption of traceability approaches in industrial settings. This paper reports on two controlled experiments performed with different participants to investigate whether the use of architecture traceability can significantly support architecture-level understanding activities. The replications with different participants allowed us to investigate whether the participants' experience plays a significant role in the understanding of software architectures with or without traceability information. In particular, we designed twelve typical questions aimed at gaining an architecture-level understanding of a representative subject system and measured how a control group (provided with no traceability information) and an experiment group (provided with traceability information) answered these questions in terms of the solutions' correctness and the participants' experience. Our findings show that the correctness of the answers of the participants in the experiment group is significantly higher than in the control group, whereas no significant differences with regard to the experience of the participants are observed.**

**Keywords: Software architecture, Architecture traceability, Architecture understanding, Empirical software engineering, Controlled experiment.**

## I. Introduction

Architects and designers require a deep understanding of the software architecture in terms of structure, behaviour, key properties, and relationships with other development artifacts, such as the requirements, detailed designs, architectural knowledge, the code base, and the documentation in order to quality control and maintain the system. An important problem in this context is that the relationships between software architecture and other development artifacts need to be evolved and maintained as the system evolves. Traceability links between the software architecture and other software development artefacts are a solution often suggested to establish consistency and maintain it over time in the face of software evolution (see, for example, [1], [2], [3], [4], [5], [6], [7]). However, there is little published empirical evidence on the usefulness of architecture traceability. To date, only one empirical study on architecture traceability has been published [8]. The study, however, involves only a limited number of participants and shows the usefulness only in the context of Compendium, a tool to visualize architecture design decisions and their rationale. It is crucial to conduct more empirical studies in the area of architecture traceability to investigate 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 the use of architecture traceability can significantly support understanding at the architectural level. Specifically, we intend to answer the following two research questions: (i) Is the correctness of answers to questions regarding the correct understanding of a software system significantly higher if traceability links between architectural models and the source code are used? (ii) Does the participants' experience have a significant interaction with the use of traceability links that affects the architecture-level understanding of a software system?

To answer the research questions, we conducted two controlled experiments at the University of Vienna, Austria. The participants in the first execution (conducted in the context of an industry training on software architecture) were 10 industrial practitioners and 7 researchers; the second execution is conducted with 92 students of the software architecture course. The replication with different participants allowed us to investigate whether the participants' experience plays a significant role in the understanding of the architecture with or without traceability links. The participants were asked to answer twelve questions aimed at gaining an understanding of a representative subject system. In the construction of these questions, nine principal understanding activities [9] that are typically performed during real-world software understanding are applied. In both experiments, half of the participants were assigned to the control group (without traceability links), while the other half was assigned to the experiment group (with traceability links). The results of the experiments were handed over to two independent researchers (later also referred to as analysts). They were asked to apply the pre-existing solution model to the recovered participants' solutions, to measure how the control groups and experiment groups answered these questions in terms of the solution correctness and participants' experience.

The rest of this paper is organized as follows: Section II discusses the related work. Section III describes the design of the controlled experiment including the introduction of variables and hypotheses, while the following Section IV

presents the details relating to the execution of the experiment. Section V presents hypotheses tests and results of the study. Section VI contains an interpretation of our findings and a discussion of threats to the validity. Section VII concludes the study and discusses future work.

## II. Related Work

As mentioned in the introduction, there exist only one earlier study by Shahin et al. [8] that provides quantitative evidence of the added value of traceability in architecture-level software understanding. We therefore also discuss other closely related controlled experiments on traceability between software development artifacts.

The works by Shahin et al. [10] and Cornelissen et al. [11] evaluate the support given by traceability visualization tools. Shahin et al. [10] investigate the usefulness of Compendium [8], a tool to visualize architectural design decisions and their rationale, as a kind of traceability information. The experiment was conducted with 10 participants, who had to understand the existing design and to make the new design according the new requirement, with and without Compendium tool. The results indicate 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.

Cornelissen et al. [11] investigate the usefulness of ExTraVis [12], which offers two interactive views: the sequence view and the circular bundle view of large execution traces, for program comprehension. The experiment involves 32 participants. The participants were asked to perform eight typical tasks aimed at gaining an understanding of a representative subject system, and measured how a control group (using the Eclipse IDE) and an experiment group (using both Eclipse and ExTraVis) performed these tasks in terms of time spent and solution correctness. The results show that ExTraVis significantly decreases the time requirements and increases the correctness for program comprehension.

Mader and Egyed [13] investigate the usefulness on traceability links between requirements and the source code. Mader and Egyed conducted a controlled experiment with 52 students of computer science. The students were asked to perform eight maintenance tasks, half of the tasks with and the other half without traceability information, on two third-party development projects [13]. Their findings show that requirement traceability saves downstream cost and can profoundly improve software maintenance quality.

De Lucia et al. [14], [15] investigate the usefulness of Information Retrieval (IR) based traceability recovery tools, also focussing on traceability links between requirements and the source code. In a first experiment [14], subjects had to perform two traceability recovery tasks, with and without COCONUT tool, on a development project carried out with 16 master students. Their results show that COCONUT significantly helps to improve the similarity between source code and related requirements in presence of comments, while a practical improvement was also seen without considering comments. In a second study, De Lucia et al. [15] conducted a controlled experiment and a replication to investigate the usefulness of traceability between use cases and the source code, and between interaction diagrams and the test cases. The experiment involved 32 master students who had to perform two traceability recovery tasks, with and without an IR-based traceability tool, ADAMS Re-Trace [16], on a software repository of a completed project. The results demonstrate that the use of a IR-based traceability tool significantly reduces the time spent on recovering trace links and improves the tracing accuracy of the software engineer.

Qusef et al. also conducted a controlled experiment and its replication to investigate the support provided by SCOTCH [17], another IR-based traceability recovery tool, focussing on traceability recovery between tests and the source code [18]. The experiment was conducted with 32 Bachelors students. The students were asked to perform two traceability recovery tasks on two systems to identify the classes tested by a set of JUnit tests. The results indicate that SCOTCH provides better support to a software engineer, compared with two existing tools (NC, based on naming conventions, and LCBA, based on the last call before an assert), to identify a higher number of correct links with higher accuracy.

None of the earlier studies in the literature provides quantitative evidence of the added value of traceability links between architecture models and the source code. Furthermore, 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 this experiment, for practical reasons and to study the foundational concepts rather than a specific tool, the design of the software and traceability links were readily provided in a printed documents, to investigate the support provided by traceability links between architectural models and more low-level artifacts (here: the source code) for architecture-level understanding of the software system, rather than the support provided by a specific tool.

## III. Design of the Experiment

For the design of the experiment, the guidelines for experiments' conduct by Kitchenham et al. [19] and Wohlin et al. [20], and reporting guidelines by Jedlitschka and Pfahl [21] were used. Kitchenham et al. present general guidelines for experimentation in software engineering and give some advices concerning the context, design, data collection, analysis, presentation, and interpretation of empirical studies without going into detail. They were primarily used in the planning phase of this experiment. Wohlin et al. present the experiment phases in more depth, and also discuss statistical tests and their suitability for different kinds of studies. Their guidelines were used as a reference for the analysis and interpretation of the results. The guidelines for reporting controlled experiments by Jedlitschka and Pfahl are used to describe the experiment in this paper. Note that the subsequent 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 II; inferences and lessons learned are discussed in Section VI; interpretation and general limitations of the study are described in Section VI-B. The usage of this reporting template, however, introduces a certain level of redundancy, because a distinction between the design and the actual execution of the experiment

is made. Some subsections of the design of the experiment (Section III) are similar to the execution (Section IV).

### A. Goal, hypotheses, parameters, and variables

The goal of this experiment is to empirically investigate the support provided by traceability links between architectural models and more low-level artifacts (here: the source code) for architecture-level understanding of the software system by developers and architects with different levels of experiences. The quality focus is on ensuring high understandability. The perspective taken in this study is both of researchers, investigating how effective the traceability links are in understanding the software system for developers and architects with different levels of expertise, and of architects or project managers, investigating the possibility of adopting the traceability links within his/her organization, depending on the skills of the involved human resources. The research goal led to the following null hypotheses and corresponding alternative hypotheses:

$H_{O1}$: The use of traceability links between architectural models and the source code *does not significantly improve* the architecture-level understanding of the software system.

$H_1$: The use of traceability links between architectural models and the source code *significantly improves* the architecture-level understanding of the software system.

$H_{O2}$: The participants' experience *significantly interacts* with the use of traceability links between architectural models and the source code to impact the architecture-level understanding of the software system.

$H_2$: The participants' experience *does not significantly interact* with the use of traceability links between architectural models and the source code to impact the architecture-level understanding of the software system.

*1) Dependent variables:* The goal of the experiment is to find out, if traceability links between architectural models and more low-level artefacts (here: the source code) significantly improve the architecture-level understanding of a software system by developers and architects with different levels of experiences. Therefore, two different treatments were defined for the participants. One group of participants was explicitly told to answer the twelve questions aimed at gaining an architecture-level understanding of a representative subject system by using the information from the architectural documentation and the source code of the system. The participants in the other group performed the same task, but additionally received the traceability links between architectural models and the source code. The first group is referred to as the control group, the latter as the experiment group.

The dependent variable in this study was the correctness of recovered answers, as shown in the Table I. Note that half of the questions in the experiment were designed as objective, multiple-choice questions, while the other half was designed as subjective, open-ended questions. In the construction of these questions, nine principal understanding activities [9] that are typically performed during real-world software understanding are applied. The objective questions (of type multiple-choice) were designed in the experiment with four possible options, which includes the correct answer and three plausible wrong answers. They were designed to help participants' in performing the typical understanding tasks within the experiment session (maximum 90 minutes) and to make it easier to abstract results. The correctness of objective questions was measured using zero and one scales. The correctness of subjective (or open-ended) questions was accessed by using recall, precision, and F-measure, the standard metrics used to evaluate the performances of information retrieval systems [22], [23]. This was primarily done to objectively evaluate the correctness of subjective questions rather than by intuitive or ad-hoc human measures. The level of scaling for information retrieval measures falls in the interval $[0, 1]$. Because answers to the subjective questions consists of a list of system elements (e.g., architectural components, source code classes), the following aspects are taken into consideration to calculate the information retrieval statistics:

- The set of correct elements expected in the solution to question i ($C_i$).
- The set of elements mentioned in the solution to question i by participant p ($M_{p,i}$).

Based on the above definition, recall and precision are computed for every subjective question. Recall is the percentage of correct matches retrieved by an experiment subject, while precision is the percentage of retrieved matches that are actually correct.

$$Recall_{p,i} = \frac{\mid M_{p,i} \cap C_i \mid}{C_i} \qquad Precision_{p,i} = \frac{\mid M_{p,i} \cap C_i \mid}{M_{p,i}}$$

Because recall and precision measure two different concepts, it can be difficult to balance between them. We used F-measure, a standard combination of recall and precision, defined as their harmonic mean, to measure the global correctness of subjective questions from the experiment participants.

$$F - measure_{p,i} = 2 * \frac{precision_{p,i} * recall_{p,i}}{precision_{p,i} + recall_{p,i}}$$

| Description | Scale Type | Unit | Range |
|---|---|---|---|
| Correctness of recovered answers | Interval | Points | [0, 1] |

TABLE I. DEPENDENT VARIABLE

| 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, 1–3, 3–7, >8 |
| Architecture experience | Ordinal | Years | 4 classes: 0, 1–3, 3–7, >8 |
| Affiliation | Nominal | N/A | Academia, Industry, Other |

TABLE II. INDEPENDENT VARIABLES

*2) Independent variables:* Five independent variables were observed during the experiment, as shown in Table II. They concern the personal information (programming experience, architecture experience, affiliation), group affiliation (control group or experiment group) and time spent in the experiment. These variable could have an influence on the dependent variables, which is eliminated by balancing the characteristics between the control group and the experiment group.

### B. Experiment Design

To test the hypotheses, two executions of a controlled experiment [24] are conducted at the University of Vienna, Austria using exactly the same design. The experiment executions were conducted as practical sessions on software architecture understanding.

*1) Participants:* The participants in the first execution were 10 industrial practitioners and 7 researchers, and it was conducted in the context of an industry training on software architecture performed by our research group. The second execution is conducted with 92 students of the software architecture course held at University of Vienna. All the participants had knowledge of software development and software architecture, as well as of software traceability.

*2) Object:* The software system to be architecturally understood by participants was the Soomla Android store[1] Version 2.0, a framework for supporting virtual economy in mobile games. It allows mobile game developers to easier implement virtual currencies (tokens, coins, gems, etc.), virtual goods, and in-app purchases. The choice of using this particular object is motivated by the following factors:

- The Soomla Android store is a free open source framework, which enables us to conduct the experiment and disseminate its results.
- It is used in real-world games; that is, it has industrial relevance.
- It addresses a well-known application domain that is likely known to the subjects from familiar real-world game applications.
- The Soomla Android store is written in the Java programming language with which the participants were sufficiently familiar.
- The source code of the Soomla Android store adheres to coding standards and is rather easy to understand for most potential subjects.
- The overall source code of the Soomla Android store comprises of 54 source code classes distributed across 8 packages, containing a total of 3623 KLOC (excluding blank lines and commented lines); that is, it is comprehensible for the participants within an experiment session, but not too simple.
- The experimenters were familiar with the internals of the Soomla Android store.

*3) Blocking:* To be able to explicitly analyse the influence of traceability links between architectural models and more low-level artefacts (here: the source code) in architecture-level understanding of a software system, the participants were split into two balanced groups. One group of participants was asked to answer the architecture-level software understanding questions by using the information from the architectural documentation and the source code of the system. The participants of the other group performed the same task, 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.

Because of rather small sample size in the first execution, it was decided to assign the participants into two balanced groups based on their programming experience (0 years, 1-3 years, 3-7 years, 8 or more years), architecture experience (0 years, 1-3 years, 3-7 years, 8 or more years) and affiliation (university, industry, other). As the sample size in the second execution was sufficiently large, the participants were randomly assigned to the two groups.

*4) Instrumentation:* To obtain the necessary data related to the influence of traceability links between architectural models and the source code in architecture-level understanding of the software system, the instruments discussed in the following paragraphs were used to carry out the experiment.

*a) Two pages of architectural documentation about the Soomla Android store version 2.0:* 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.

*b) A colourized copy of the source code for the Soomla Android store:* The cover page alphabetically lists the code class, their respective page number(s) and owned source code package name. The subsequent pages lists the syntax-highlighted source code of the system. Note that the participants in the experiment group were also provided with the list of traceability links, showing the relations between architectural components and their realized code classes.

*c) A questionnaire to be filled-in by the participants during the experiment:* 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 understanding questions. In the context of these questions, two important criteria are applied: (i) the questions should be representative for key understanding and maintenance contexts, and (ii) they should be imaginatively constructed to measure the deeper understanding from participant groups. In this regard, nine principal understanding activities that are typically performed during real-world software understanding are applied. Please refer to [9] for the detailed description of these activities. Guided by these activities, 12 representative questions (shown in Table III) are defined that highlight many of the Soomla Android store aspects at both a high-level of abstraction (architecture-level) and a low-level of abstraction (source-code-level).

Note that the first six questions were designed as objective, multiple-choice questions to help participants' in answering the understanding questions within the experiment session (maximum 90 minutes) and to make it easier to abstract results. The multiple choice questions in the experiment were designed with four possible options, which includes the correct answer and three plausible wrong answers. While latter six questions were designed as subjective, open-ended questions to represent typical maintenance situations, as their answers consists of a list of system elements (e.g., architectural components, source code classes).

[1] see: `http://project.soom.la/`

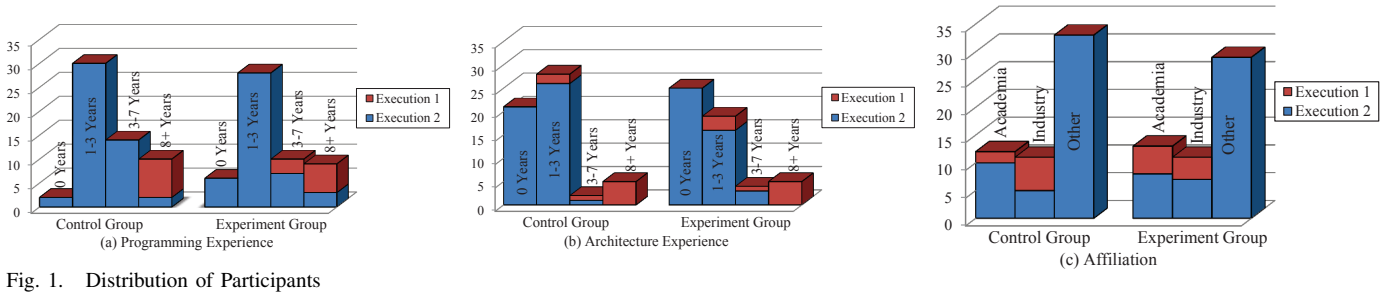


Fig. 1. Distribution of Participants

| ID | Description |
|---|---|
| Q1 | Identifying the main class for triggering and orchestrating the key functions of the Soomla Android store |
| Q2 | Identifying the primary class for manipulating the storage and retrieval operations in the database |
| Q3 | Investigating the realization relationship between *StoreController.java* class (implementation) and component in the architecture (specification) |
| Q4 | Investigating the component in the architecture that provides the security at runtime |
| Q5 | Understanding how encryption and decryption keys are stored in the database |
| Q6 | Identifying the architectural component(s) that raise and use the events related to the store assets and their purchasing |
| Q7 | Identifying the classes that implement store assets and their categorization (types) |
| Q8 | Identifying the classes that implement pricing models for store assets |
| Q9 | Determining the classes that implements encryption and decryption functionality |
| Q10 | Investigating the impact of migrating the database from *SQLLiteDatabase* to the *SQLServerDatabase* |
| Q11 | Investigating the impact of adding to or changing the functionality of implemented billing framework |
| Q12 | Listing the strongly coupled classes to the *StorageManager.java* class |

TABLE III. QUESTIONNAIRE FOR ARCHITECTURE-LEVEL SOFTWARE UNDERSTANDING

*5) Blinding:* To eliminate subjective bias on the part of both experimental participants and the experimenters, double blinding was applied in the experiment. Although, participants perceived that there are two different groups, they were not aware about the purpose of group division and their group affiliation. The results of the experiment were handed over to two independent researchers who did not know the real identity of the participants. This was done to prevent the experiment from being biased. To be able to compute the correctness of recovered answers, the researchers were asked to match the participants' answers with the original solution model. The correctness of objective questions was measured using zero = false and one = true scales; the correctness of subjective questions was measured using information retrieval statistics, ranges from zero worst to one best correctness. The nature of evaluations for objective and subjective questions allows analysts to objectively evaluate the correctness of recovered answers rather than by intuitive or ad-hoc human measures.

*6) Data collection procedure:* After introduction and grouping, the participants received the instruments, mentioned in Section III-B4. The provided instruments had to be used to answer the questionnaire. The participants were distributed over separate rooms according to their group membership. At least one experimenter was present in each room during the whole duration, enabling the participants to pose clarification questions and restrict them from consulting others and using forbidden material. The participants were given maximum 90 minutes to individually answered the questionnaire. After completion of session, the filled-in questionnaire 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.

The discussion questions were concerning difficulties in the experiment and impact of traceability links in answering the questionnaire. While this information is not required for testing the hypotheses, it is used for validation and interpretation of the results. The questions after the experiment are asked for three reasons: 1) because the traceability links between architectural models and the source code could have influenced the participants of the control group prior to the experiment; 2) because we are interested in a way the participants answered the understanding questionnaire; 3) in the light of discussion, the participants of the control group could have guessed that traceability links play an important role in the other group, and consciously or unconsciously also focused on the traceability links.

## IV. EXECUTION

### A. Sample and Preparation

As described in Section III-B, the experiment was conducted in two practical sessions on software architecture understanding at the University of Vienna, Austria, one in the context of an industry training, the other in the context of a lecture on software architecture. The first session took place with 17 experienced participants; the second session was conducted with 92 students of the software architecture 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 from the two executions of the experiment, but also shows the separate data of the executions. The Sub-figures (a) and (b) show the previous experience of the participants concerning programming and architecture, while Sub-figure (c) shows the affiliation of the participants. Note that the previous experience of the participants in the first execution of the experiment is quite comparable among the control group and the experiment group. As mentioned earlier, the participants belonging to the first execution of the experiment had been manually assigned to the two balanced groups due to the small sample size. The second execution of the experiment have slightly more participants with longer experience in the control group. This is probably because participants of the second execution were randomly assigned to the two groups due to the rather sufficient sample size. However, overall the experiences and affiliations

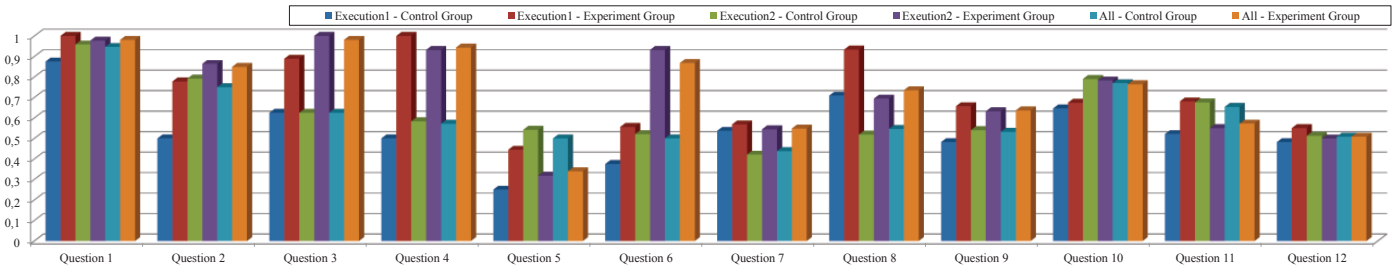


Fig. 2. Average correctness for each experiment question

are rather well balanced both in the two executions and in total.

### B. Data collection performed

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

### C. Validity procedure

This experiment was conducted in a controlled environment. The participants were assigned to the 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 pose 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 experiment session. No unexpected situation occurred during the experiment.

## V. Analysis

### A. Descriptive statistics

This section shows the results of the experiment as a first step in the analysis. The first subsection V-A1 concerns the influence of traceability links on the architecture-level understanding of the software system. The latter subsection V-A2 presents an analysis of the influence of other factors, mainly related to the influence of the participants' experience on the architecture-level understanding of the software system.

| Execution | Group Affiliation | Participants | Mean | Median | Std. Dev. |
|---|---|---|---|---|---|
| Execution 1 | Control Group | 8 | 6.501064 | 5.925 | 2.09378 |
| | Experiment Group | 9 | 8.730342 | 9.008081 | 1.998951 |
| Execution 2 | Control Group | 48 | 7.479229 | 7.63254 | 1.790458 |
| | Experiment Group | 44 | 8.725067 | 9.0663 | 1.474313 |
| All | Control Group | 56 | 7.339491 | 7.561292 | 1.84843 |
| | Experiment Group | 53 | 8.725963 | 9.065934 | 1.553107 |

TABLE IV. Descriptive Statistics for the influence of method

*1) Influence of traceability links:* Table IV shows the comparisons for the control groups and the experiment groups from the two executions of the experiment as well as for the aggregated data obtained by merging the data from both experiments. The results in the table are based on the sum of the results of the questions for each participant. In total, the correctness of answers in the experiment groups is higher than the correctness of answers in the control groups.

The data in Figure 2 reports average correctness for each experiment question from both executions of the experiment as well as the aggregated data from both experiments. The figure shows that the participants of the experiment group belonging to the first execution of the experiment have a higher average correctness for all questions than the control group. However, the participants of the control group of the second execution of the experiment outperformed the participants of the experiment group in Question 5 and 11.

An important reason for this result in Question 5, which is of type multiple-choice question, is that more of the participants of the experiment group got confused by a strongly crafted distracter. However, for this question both of groups performed rather poorly. The reason for this result in Question 11, which is of type open-ended question, might be that the participants in the experiment group did not exactly know what to look for in the traceability links, whereas the participants of the control group probably took one of the relevant classes as the starting point and followed the import statement(s) in other classes to identify the answer to the question.

*2) Influence of Participants' Experience:* The participants' in the first execution of the experiment had several years of programming and architecture experience: they were either affiliated with the academia or industry. While the participants in the second execution of the experiment were students of the software architecture course, many of the students work either for the academia and industry. For this reason, the influence of participants' previous experience - '0 years' vs. '1-8+ years' on the method is measured for both executions of the experiment. The participants with '0 years' of architecture experience were considered as novices, while other participants with '1-8+ years' of architecture experience were considered as experienced in the experiment.

| Group Affiliation | Arch-Experience | Participants | Mean | Median | Std. Dev. |
|---|---|---|---|---|---|
| Control Group | 0 years | 21 | 7.177572 | 7.009244 | 1.903803 |
| | 1 – 8 years | 35 | 7.436643 | 7.960318 | 1.835499 |
| Experiment Group | 0 years | 25 | 8.524966 | 9.065934 | 1.499115 |
| | 1 – 8 years | 28 | 8.905424 | 9.116162 | 1.605353 |

TABLE V. Influence of participants' experience on method: Descriptive Statistics

Table V shows the descriptive statistics for the influence of participants' experience on the correctness of recovered answers, grouped by participants' experience and group affiliation. The results show slight differences in terms of the

experience of participants. That is, the participants with '1-8+ years' of architecture experience in the experiment group have slightly better level of understanding than the participants with '0 years' of architecture experience.

### B. 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.

Two of the participants from the second execution of the experiment did not answer all the questions. In particular, one participant in the control group did not answer six questions, and another participant in the experiment group did not answer one question. This result in seven missing data points in the experiment. As it seems that these participants had 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 average correctness for the questions for each participant. Note that two of the participants from the experiment group reached a considerable lower correctness than the other members of this group. A closer analysis showed that they could not properly make use of traceability links to answer the understanding questions. However, their correctness were not excluded as outliers, because the difference to the other participants is not strong enough. Excluding the data points would have introduced a potential vulnerability of the study results.

### C. 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 both groups from the two executions of the experiment had acknowledged that they had enough time to answer the questionnaire. However, the participants of the control groups experienced more difficulties in answering the questionnaire than the participants of the experiment group. The same happened also for the experience of the participants: The participants with '0 years' of experience encountered more difficulties than the participants with '1-8+ years' of experience, while no significant differences emerged between correctness of the recovered answers.

The participants were also asked about their familiarity with the application domain. The answers imply that mobile game economies, which is the application domain of the Soomla Android store, is well-known to the participants from familiar real-world game applications. No noticeable differences emerged between participants and between different questions regarding the application domain.

The next two questions concerned the usage and helpfulness of traceability links for architecture-level understanding of the software system. First, the participants were asked whether traceability links are useful to architecturally comprehend the software system. 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 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 the traceability links for software understanding.

Finally, the participants were asked to briefly describe how the questions were answered. This was primarily done to confirm that the experiment group used the traceability links and to find out if the control group used any other systematic way to architecturally understood the software system. The answers of the control group imply the focus on an intuitive approach, which was mainly driven by personal experience or judgments. The respondents stated that they answered the questions 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 group show a focus on the traceability links. The respondents of experiment group stated that they used traceability links to identify the architecture artefacts in the code classes and vice versa. They confirmed that they primarily used this additional knowledge for giving the answers to the questions.

### D. Hypothesis testing and results

*1) Influence of traceability links:* To be able to test the first null hypothesis $H_{O1}$, the influence of traceability links on the correctness of participants' answers is measured. In the analysis of the experiment, the Shapiro-Wilk normality test [25] and Wilcoxon Rank-Sum test [26] 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%).

The null hypothesis $H_O$ for the Shapiro-Wilk normality test states that the input data are normally distributed. Table VI shows the results from the Shapiro-Wilk normality test for the control groups and the experiment groups. The results in the table show that the p-values of the experiment group in the second execution of the experiment and control group for aggregated data from both experiments is lower than 0.05, the null hypothesis must be rejected for these two groups. As a consequence, we cannot apply parametric statistical tests like the t-test, but should instead use non-parametric tests.

| Execution | Group Affiliation | Shapiro-Wilk Normality Test |
|---|---|---|
| Execution 1 | Control Group | W = 0.8928, p-value = 0.2487 |
| | Experiment Group | W = 0.9472, p-value = 0.6595 |
| Execution 2 | Control Group | W = 0.9615, p-value = 0.1165 |
| | Experiment Group | W = 0.9218, **p-value = 0.005454** |
| All | Control Group | W = 0.957, **p-value = 0.04438** |
| | Experiment Group | W = 0.9604, p-value = 0.07656 |

TABLE VI. RESULTS OF THE SHAPIRO-WILK NORMALITY TEST

Because both executions of the experiment were organized as longitudinal studies where each participant answered the

questionnaire, with the two possible treatments for method (no traceability links, with traceability links), it is possible to use a Wilcoxon rank-sum test to measure the differences exhibited by each participant for the two treatments. Table VII shows the results of the Wilcoxon rank-sum test for the control group and the experiment group. The table shows that the both executions of the experiment (Execution 1 and Execution 2) as well as the aggregated data from both experiments (All) provides strong evidence that $H_{O1}$ can be rejected. This means that in our two experiments the use of traceability links significantly improved the architecture-level understanding of the software system. Note that there is a noticeable difference in the p-values between the two executions of the experiment and the aggregated data from both experiments. The main reason behind this difference probably is the limited number of participants in the first execution of the experiment.

| Execution | Factor | Wilcoxon Rank-Sum Test |
|---|---|---|
| Execution 1 | Control Group vs. Experiment Group | W = 14, **p-value = 0.03595** |
| Execution 2 | Control Group vs. Experiment Group | W = 622, **p-value = 0.000703** |
| All | Control Group vs. Experiment Group | W = 844, **p-value = 0.0001057** |

TABLE VII. RESULTS OF THE WILCOXON RANK-SUM TEST

*2) Influence of participants' experience:* Hypothesis $H_{O2}$ was also evaluated with a Wilcoxon rank-sum test. The results are shown in Table VIII. Although slight differences in terms of the experience of participants' can be observed, the results can not be said to be statistically significant. This means that the hypotheses $H_{O2}$ can not be rejected.

| Group Affiliation | Factor | Wilcoxon Rank-Sum Test |
|---|---|---|
| Control Group | Novices vs. Experienced | W=350, p-value=0.7736 |
| Experiment Group | Novices vs. Experienced | W=300, p-value=0.3809 |

TABLE VIII. RESULTS OF THE WILCOXON RANK-SUM TEST

## VI. INTERPRETATION

### A. Evaluation of results and implications

*1) Influence of traceability links:* Hypotheses $H_{O1}$ and $H_1$ concern the influence of traceability links. The results pointed out in Section V provides strong evidence that the null-hypothesis $H_{O1}$ can be rejected. Thus, according to our experiments, there is evidence that the architecture-level understanding of a software system is higher if the traceability links between architectural models and the source code are used.

*2) Influence of participants' experience:* To be able to test the second null hypothesis $H_{O2}$, the influence of the participants experience on the correctness of recovered answers is measured. Note that this analysis was performed on the whole dataset, similar to the study by Wohlin et al. [20], who compared the software development process using C and C++. The parametric statistics could be applied at the dataset, since (i) the experiment design, instrumentation and procedure is exactly the same, (ii) the way participants' experience has been evaluated is the same. It should be noted that only the substantial difference, i.e. experience of the participants, is considered as an experimental factor.

Hypotheses $H_{O2}$ and $H_2$ concern the influence of participants experience. The results do not provide evidence to conform or reject the null hypothesis. Thus we are unable to show that the participants' experience has a significant interaction with the use of traceability links to effect the architecture-level understanding of the software system. This result is surprising to us because we have assumed that understanding of recovered solutions for experienced participants is significantly higher than for novice participants.

### B. Limitations of the study

Several levels of validity threats have to be considered in the experiment. We have considered the classification scheme for validity in experiments by Cook and Campbell [27]. 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. All validity threats in the experiment are categorized based on this classification.

*1) Internal validity:*

- The participants' population in the experiment might not be sufficiently competent. This might influence the results of the experiments. In this particular experiment, all the participants' from the two executions of the experiment had knowledge about software development and software architecture, as well as of software traceability. They all studied the previous lectures of at least the software architecture course.
- The variation in human performance might distort the results of the experiments, and then the performance differences would not arise from the difference in treatments. In this particular experiment, the participants' experience is quite comparable among the control group and the experiment group in both executions of the experiment (as shown in Fig. I). Thus, this factor is not seen as a strong threat to validity.
- The experiment was conducted in a controlled environment in separate rooms under supervision of at least one experimenter. Although, it was 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 understanding of the questionnaire could have been biased towards the experiment group. The threat, however, is mitigated by applying an established task framework [9] to ensure that many aspects of typical understanding contexts are covered. As a result, the questionnaire concerned both global and detailed knowledge, as well as static and dynamic aspects. Therefore, we do not consider it a highly relevant threat to validity.
- The analysts in the experiment could have graded the recovered answers incorrectly. We tried to mitigate

this risk by providing the original solution model to the analysts. The analysts were asked to applying the solution model to the recovered participants' solutions. The solution model clearly states the correct element(s) for each question. Furthermore, the results have also been verified by the authors of this paper.

- Finally, the analysts could have been biased towards the experiment group. 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.

*2) External validity:*

- As discussed in Section III-B, the first execution of the experiment is conducted with experienced participants. They all have an academic or industrial background in several software engineering disciplines and a strong interest in software architecture and software traceability. The second execution of the experiment was conducted with rather inexperienced participants, the students of a software architecture course. Nevertheless, the results of our study imply that the participants' experience does not have an significant influence on the external validity of results. Therefore, we conclude that the lower level of experience of the participants in the second execution of the experiment does not distort the study results.
- The instrumentation in the experiment might have been unrealistic or old-fashioned. In this case, the traceability recovery was based on a printed traceability links. In practice, different tools would be used to support traceability recovery. These tools are primarily used to formulate and maintain the traceability links. In this experiment, for practical reasons and to study the foundational concepts rather than a specific tool, the design of the software and traceability links were readily provided in a printed document. We assume that the measured effect of a experiment group 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.

*3) Construct validity:*

- The fact that only one object (the Soomla Andorid store) is used in the experiment, introduces the risk that the cause construct is under-represented. Theoretically, the results could look different if software system with higher number of code classes and multiple architecture views would be used for the understanding. We assume that the used system is representative for large and medium-size object-oriented systems. The threat, however, cannot totally be ignored.
- Another potential threat to validity is the interactions between experiment groups. We tried to mitigate this risk by the design of the experiment that allowed to separate the analysis of the different factors and of their interactions. Thus, this factor is not seen as a strong threat to validity.

*4) Conclusion validity:*

- A threat to validity might result from the use of objective, multiple-choice questions because of poorly crafted distracters (wrong options) and guessing. We tried to mitigate this risk by designing the multiple-choice questions with four possible options, which includes the correct answer and three distracters (plausible wrong answers): The distracters represent common misapprehensions and/or mistakes within a discipline and require appropriate decision making methodologies to interpret them. Nevertheless, the multiple-choice questions were designed in the experiment to help participants' in performing the typical understanding tasks within the experiment session (maximum 90 minutes) and to make it easier to abstract results. All these threats, however, cannot totally be ignored.
- Another threat to validity might result from the interpretation of the subjective questions because answers to the subjective questions consists of a list of system elements (e.g., architectural components, source code classes). We mitigated this risk by calculating the standard information retrieval metrics for recovered subjective solutions from all questions. We argue that information retrieval measures allow analysts to objectively evaluate the correctness of subjective questions 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 experiment, 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 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.

## VII. Conclusions and future work

In this paper, we describe the results of two controlled experiments that were conducted to find out if traceability links between architecture models and the source code are beneficial for architecture-level understanding of the software system. The experiments focus on the correctness of the recovered answers and the participants' experience. The results of the experiments shows that providing traceability links between architectural models and the code base leads to a higher level understanding of the software architecture. According to our results, the experience of participants did not play a significant role in linking architectural models and the code base.

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 replicate the experiment with a traceability tool.

## VIII. 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.

## References

[1] J. Han, "Experience with designing a requirements and architecture management tool," in *Proceedings of the International Conference on software Methods and Tools (SMT'00)*, ser. SMT '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 179–. [Online]. Available: http://dl.acm.org/citation.cfm?id=555920.830363

[2] G. A. A. Cysneiros, F. Andrea, and Z. G. Spanoudakis, "Traceability approach for i* and uml models," in *in Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS03*, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.1309

[3] A. Tang and J. Han, "Architecture rationalization: A methodology for architecture verifiability, traceability and completeness," in *Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, ser. ECBS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 135–144. [Online]. Available: http://dx.doi.org/10.1109/ECBS.2005.17

[4] H. Tran, U. Zdun, and S. Dustdar, "Vbtrace: using view-based and model-driven development to support traceability in process-driven soas," *Softw. Syst. Model.*, vol. 10, no. 1, pp. 5–29, Feb. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10270-009-0137-0

[5] G. Buchgeher and R. Weinreich, "Automatic tracing of decisions to architecture and implementation," in *Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, ser. WICSA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 46–55. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2011.16

[6] N. Ubayashi and Y. Kamei, "Architectural point mapping for design traceability," in *Proceedings of the eleventh workshop on Foundations of Aspect-Oriented Languages*, ser. FOAL '12. New York, NY, USA: ACM, 2012, pp. 39–44. [Online]. Available: http://doi.acm.org/10.1145/2162010.2162022

[7] L. G. P. Murta, A. van der Hoek, and C. M. L. Werner, "Archtrace: Policy-based support for managing evolving architecture-to-implementation traceability links," in *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 135–144. [Online]. Available: http://dx.doi.org/10.1109/ASE.2006.16

[8] 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*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 2367–2368. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774577

[9] M. J. Pacione, M. Roper, and M. Wood, "A novel software visualisation model to support software comprehension," in *Proceedings of the 11th Working Conference on Reverse Engineering*, ser. WCRE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 70–79. [Online]. Available: http://dl.acm.org/citation.cfm?id=1038267.1039039

[10] 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*, ser. ECSA '11. New York, NY, USA: ACM, 2011, pp. 2:1–2:8. [Online]. Available: http://doi.acm.org/10.1145/2031759.2031762

[11] B. Cornelissen, A. Zaidman, and A. van Deursen, "A controlled experiment for program comprehension through trace visualization," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 341–355, 2011. [Online]. Available: http://dx.doi.org/10.1109/TSE.2010.47

[12] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. van Wijk, and A. van Deursen, "Understanding execution traces using massive sequence and circular bundle views," in *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on*, 2007, pp. 49–58. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2007.39

[13] P. Mader and A. Egyed, "Assessing the effect of requirements traceability for software maintenance," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, 2012, pp. 171–180. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2012.6405269

[14] A. De Lucia, R. Oliveto, F. Zurolo, and M. Di Penta, "Improving comprehensibility of source code via traceability information: a controlled experiment," in *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ser. ICPC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 317–326. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2006.28

[15] A. De Lucia, R. Oliveto, and G. Tortora, "Assessing ir-based traceability recovery tools through controlled experiments," *Empirical Softw. Engg.*, vol. 14, no. 1, pp. 57–92, Feb. 2009. [Online]. Available: http://dx.doi.org/10.1007/s10664-008-9090-8

[16] ——, "Adams re-trace: traceability link recovery via latent semantic indexing," in *Proceedings of the 30th international conference on Software engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 839–842. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368216

[17] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley, "Scotch: Test-to-code traceability using slicing and conceptual coupling," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, 2011, pp. 63–72. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2011.6080773

[18] A. Qusef, G. Bavota, R. Oliveto, A. D. Lucia, and D. Binkley, "Evaluating test-to-code traceability recovery methods through controlled experiments," *Journal of Software: Evolution and Process*, pp. n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/smr.1573

[19] 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*, vol. 28, no. 8, pp. 721–734, Aug. 2002. [Online]. Available: http://dx.doi.org/10.1109/TSE.2002.1027796

[20] C. Wohlin, *Experimentation in Software Engineering: An Introduction: An Introduction*, ser. The Kluwer International Series in Software Engineering. Kluwer Academic, 2000. [Online]. Available: http://books.google.es/books?id=nG2UShV0wAEC

[21] A. Jedlitschka, D. Hamann, T. Göhlert, and A. Schröder, "Adapting profes for use in an agile process: An industry experience report," in *PROFES*, 2005, pp. 502–516. [Online]. Available: http://dx.doi.org/10.1109/ISESE.2005.1541818

[22] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999. [Online]. Available: http://www.dcc.ufmg.br/irbook/

[23] D. Harman, "Information retrieval," W. B. Frakes and R. Baeza-Yates, Eds. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992, ch. Ranking algorithms, pp. 363–392. [Online]. Available: http://dl.acm.org/citation.cfm?id=129687.129701

[24] B. Boehm, V. Basili, H. Rombach, and M. Zelkowitz, *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili*. Springer, 2005. [Online]. Available: http://books.google.at/books?id=ski1Ms57pScC

[25] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. pp. 591–611, 1965. [Online]. Available: http://www.jstor.org/stable/2333709

[26] 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. [Online]. Available: http://books.google.at/books?id=9fvePgAACAAJ

[27] T. Cook and D. Campbell, *Quasi-experimentation: design & analysis issues for field settings*. Houghton Mifflin, 1979. [Online]. Available: http://books.google.at/books?id=9Up-AAAAIAAJ