DOI: 10.1007/s10664-017-9591-4

Noname manuscript No.
(will be inserted by the editor)

Do Software Models Based on the UML Aid in Source-Code Comprehensibility? Aggregating Evidence from 12 Controlled Experiments

Giuseppe Scanniello · Carmine Gravino · Marcela Genero · José A. Cruz-Lemus · Genoveffa Tortora · Michele Risi · Gabriella Dodero

Received: date / Accepted: date

Abstract In this paper, we present the results of long-term research conducted in order to study the contribution made by software models based on the Unified Modeling Language (UML) to the comprehensibility of Java source-code. We have conducted 12 controlled experiments in different experimental contexts and on different sites with participants with different levels of expertise (i.e., Bachelor's, Master's, and PhD students and software practitioners from Italy and Spain). A total of 333 observations were obtained from these experiments. The UML models in our experiments were those produced in the analysis and design phases. The models produced in the analysis phase were created with the objective of abstracting the environment in which the software will work (i.e., the problem domain), while those produced in the design phase were created with the goal of abstracting implementation aspects of the software (i.e., the solution/application domain). Source-code comprehensibility was assessed with regard to correctness of understanding, time taken to accomplish the comprehension tasks, and efficiency as regards accomplishing those tasks. In order to study the global effect of UML models on source-code comprehensibility, we aggregated results from the individual experiments using a meta-analysis. We made every effort to account for the heterogeneity of our experiments when aggregating the results obtained from them. The overall results suggest that the use of UML models affects source-code comprehensibility. Indeed, models produced in the analysis phase might reduce source-code comprehensibility, while increasing the time taken to complete comprehension tasks. That is, browsing source code and this kind of models together negatively impacts on the time taken to complete comprehension tasks without having a positive effect on the comprehensibility of source code. One plausible justification for this is that the UML models produced in the analysis phase focus on the problem domain. That is, models produced in the analysis phase say

Giuseppe Scanniello
University of Basilicata
E-mail: giuseppe.scanniello@unibas.it

Carmine Gravino, Genoveffa Tortora, Michele Risi
University of Salerno
E-mail: {gravino, totora, mrisi}@unisa.it

Marcela Genero, José A. Cruz-Lemus
University of Castilla-La Mancha
E-mail: {marcela.genero; joseantonio.cruz}@uclm.es

nothing about source code and there should be no expectation that they would, in any way, be beneficial to comprehensibility. On the other hand, UML models produced in the design phase improve source-code comprehensibility. One possible justification for this result is that models produced in the design phase are more focused on implementation details. Therefore, although the participants had more material to read and browse, this additional effort was paid back in the form of an improved comprehension of source code.

Keywords Aggregation, Heterogeneity, Unified Modeling Language, Controlled Experiments.

1 Introduction

The Unified Modeling Language (UML) [46] is considered to be the de-facto standard in the analysis, design, and evolution of object-oriented software [17,29], despite the fact that domain specific *modeling* languages are increasing in popularity [34]. However, many software companies are still reluctant to use UML because it is perceived to be difficult to learn and use [2]. It may, therefore, be important, if not crucial, to investigate whether or not the use of the UML makes a practical difference in software development and evolution, thus possibly encouraging resilient companies to adopt UML.

The UML has been the subject of a number of empirical studies in the software engineering field [11]. Of these studies, only a few are focused on the usage of this notation throughout the software development life cycle [3]. This lack is even more evident in software maintenance and evolution (e.g., [9,53]). Scanniello et al. [56] conducted an industrial survey regarding the use of the UML in the software industry. The results showed that software engineers wishing to deal with software maintenance and evolution very often have at their disposal only UML-based models (or simply UML models) produced in the requirements engineering process (or analysis phase) and, in a few cases, those produced in the design phase. Using the findings of this survey as a basis, we began long term research with the aim of studying the contribution made by UML-based models produced in the analysis and design phases to source-code comprehensibility. In particular, we conducted 12 controlled experiments with different kinds of participants [27,28,53–55]. In order to aggregate the results of these experiments and to obtain the global effect of analysis and design UML models on source-code comprehensibility, we carried out a meta-analysis on the results obtained from the individual experiments. In this paper, we present the results of this aggregation by attemping to answer the following research question:

- *Do software models produced in the analysis and design phases aid the comprehension of Java source code (assessed on the basis of the answers developers provide to questions on that code), and do these models affect the time taken to comprehend that code and the efficiency with which that comprehension occurs?*

The research work presented in this paper is based on that presented in [57], in which we showed the preliminary results obtained after the aggregation of our 12 controlled experiments. The current paper extends the previous work as follows: (i) we have considered issues related to the heterogeneity of the individual experiments in the aggregation of their results; (ii) we have considered an additional dependent variable, namely Efficiency, which is computed as the ratio between the level of comprehension achieved when performing a task and and the time required to complete it; and (iii) we have extended the discussion concerning related work, experimental results, and threats to validity.

In this new paper, we provide a detailed description of the following main contributions:

- The results of global effect of analysis and design UML models on source-code comprehensibility;
- A discussion regarding the possible practical implications of the results of our study;
- How to deal with the heterogeneity of experiments when using a meta-analysis to aggregate the results obtained from them.

This paper is organized as follows. We discuss related work in Section 2, while the background is presented in Section 3. Our long-term research is presented in Section 4, while the results obtained are shown and discussed in Section 5. In Section 5, we also discuss the practical implications of the results of our study from the perspectives of both researchers and professionals. We conclude the paper with our final remarks and future work in Section 6.

2 Related Work

In accordance with the research question stated previously, in this section we first focus on a set of works which highlights the use of UML diagrams as a means to maintain source-code. We conclude this section by presenting research work on model-based traceability, since the use of traceability links is a viable means to support the comprehension and maintenance of source code.

2.1 UML and Software Maintenance

There are two literature reviews related to the topic that should be taken into account [21, 64]. In the first place, Fernández-Sáez et al. [21] present a systematic mapping study (SMS) whose goal was to discover the empirical evidence related to the use of UML diagrams in source code maintenance and the maintenance of UML diagrams themselves. A total of 38 papers were found as a result of this SMS, including 66 empirical studies, but only two of them were specifically focused on source-code maintenance ([5,15]). Zhi et al. [64] also report another SMS but with a slightly different goal: the existing literature concerning software documentation cost, benefit and quality. Once again, the same two works ([5,15]) are directly related to source-code maintenance. These two empirical studies will be explained below, together with another set of experiments which was published after the performance of the two SMSs, and thus not included in them.

Using the aforementioned work as a basis, Dzidek et al. [15] performed an experiment with a group of 20 professionals. Half of them used UML documentation to carry out a set of modifications on a web-based system developed in Java, while the other half did not. This was done to assess whether providing UML documentation reduced the effort required to correctly implement a set of tasks regarding changes and increased the functional correctness and design quality of those changes. The results reported indicate that UML users had to spend more time, especially when the documentation had to be updated. However, the use of UML documentation was simultaneously always beneficial in terms of functional correctness, since fewer faults were introduced into the software maintained. It is also important to highlight that the no-UML group of participants had more problems as regards understanding the most complex part of the system.

Arisholm et al. [5] conducted two controlled experiments to assess whether UML documentation helped to reduce the effort required to change the source-code of a software system. The original experiment was conducted in Oslo, Norway, and the second in Ottawa, Canada. The first experiment was conducted with 20 3rd year undergraduate students, while the second was carried out with 4th year 78 undergraduate students. In both these experiments, the independent variable was: using (or not using) UML documentation. The participants' performances were measured by considering the time required to perform changes, excluding and including diagram modifications, correctness of these changes and the quality of the changed design. The most important result obtained was: UML documentation does not provide an advantage as regards time, although it helps to improve the correctness and quality achieved when solving the most complex tasks.

Leotta et al. [45] present a pilot experiment carried out to relate the level of alignment between UML documents and code and maintainers' efficiency . A group of 21 undergraduate students had to perform a set of 4 maintenance tasks on two systems using the Eclipse framework while surfing the UML documents provided (sequence and class diagrams). Although this was a pilot study, the results confirmed the general belief that a more aligned documentation is of greater assistence during maintenance tasks.

Fernández-Sáez et al. [22] report a family of experiments carried out at two different universities in Italy and Spain. The aim of this study was to assess whether the origin of UML diagrams (the design phase of a life-cycle or a reverse engineering technique) influences the maintenance of the corresponding source-code. A controlled experiment and two replications of it were performed. The authors involved a total of 149 MSc students (categorized according to their ability, which was calculated using their course grades). These students had to carry out adaptive and corrective maintenance tasks. The main finding of the work was that participants with a higher ability achieved better scores when using the diagrams with a forward design origin, whilst low ability participants got better scores when using reverse engineered diagrams. The authors provide a possible explanation for this situation by relating low levels of experience to difficulty in using (and/or understanding) UML diagrams.

The last experiment in this list (Fernández-Sáez et al. [20]) reports a family of four controlled experiments carried out by over 80 BSc and MSc students at three different universities in Italy, The Netherlands, and Spain. In this case, the goal was to analyze whether a high or a low level of detail in UML diagrams has an impact on the maintainability of source-code during a model-centric development. The maintainability of the source-code was measured by means of its understandability and modifiability, and the participants had to answer a multiple choice questionnaire to show how they had understood the system, in addition to performing a set of corrective maintenance tasks to show how it could be modified. The results of the family of experiments indicated that diagrams with a high level of details improved the understandability of the system, while those with a low level of details improved its modifiability. However, the authors indicate that, when attempting to replicate the study, the results obtained seemed to be incoherent and to have no clear tendency. After a thorough evaluation of this, they discovered that the diagrams had possibly been used incorrectly, or even not used at all, which would have led to the results obtained. They relate this to the day-by-day situation in industry, when users' false self-certainty may lead them not to use (or not to use properly) the system documentation, and eventually find themselves involved in unexpected and undesired situations as regards their maintenance duty.

Only a few evaluations of the benefits derived from the use of UML throughout the whole software development life cycle have been reported [3]. This lack is even more evident in the software maintenance phase with regard to the benefits of employing UML models in source code comprehension. In order to fill this gap, we present the results of long-term research conducted to study whether or not UML-based models (produced in both the requirements and analysis phases) aid source code comprehensibility. In particular, we aggregate and synthesize the outcomes of 12 controlled experiments in different experimental contexts and on different sites with participants who had different levels of expertise. A total of 333 observations were obtained from these experiments. Our study is currently the largest in the literature concerning the UML.

Table 1 presents a summary of the main features of these papers. The columns in the table are labelled as:

– Ref: contains the bibliographical reference to the paper.
– Goal: describes the objective of the experiment.
– Participants: presents the number of participants that took part in the experiment/s, plus their type (students, professionals, etc.).
– Independent variable: describes the variable whose effect on the dependent variables is to be studied. The values (treatments) of the independent variable are also presented.
– Dependent variables: presents the outcome variables, i.e., those affected by the changes made to the independent variables.
– Tasks: describes the tasks to be performed by the participants as part of the experiment.
– Results: shows the main findings obtained in the experiment.

Some other empirical studies in industry related to the research question of this work can also be found. For example, Scanniello et al. [56] show the results of an explorative survey carried out to investigate the state of practice as regards the use of UML in software development and maintenance. At that time, UML appeared to be the most widely used modeling option for software development and maintenance. In particular, 74% and 75% of the companies interviewed employed UML in the development and maintenance phases, respectively. The authors did not ask the remaining survey respondents which modeling languages were used as an alternative to UML. The companies interviewed stated that maintenance operations are commonly performed by practitioners who do not have a vast amount of experience. According to the study, an ordinary maintenance operation, such as making certain corrective changes, needs an effort range of between 1 and 5 persons/hour. These values are multiplied by 10 in the case of an extraordinary maintenance operation, such as perfective or adaptive changes.

Fernández-Sáez et al. [19] also present the preliminary findings of a case study whose intention was to discover whether the investment in UML is justified by the benefits (improved productivity and product quality) in software maintenance projects. They consequently focus on discovering what the cost and the payback of using UML in a software maintenance project are. They carried out a case study in a multinational company with an IT department of 800-1000 employees. They collected data from the files shared by the department and, principally, by interviewing the companys personnel (20 useful interviews). They eventually concluded that the employees reported several benefits of using UML: a better understanding of the problem domain, improved communication, a reduction in software defects, an improvement to the quality and a reduction in the software maintenance effort.

Garousi et al. [26] present a multi-method empirical approach, and include a survey, a case study and some action-research in their proposal. A company providing satellite navigation system products was changing its software development processes and intended to measure which key factors were impacting on documentation usage (information sources, life-cycle phase, document type, roles, degree of experience, and patterns of usage) and which attributes were affecting the quality of the documentation. An exploratory survey was conducted with 25 employees to assess the aforementioned factors and attributes. Its results were reused in a case study during which the company data was employed, and the conclusion was reached that the usage of documentation differed according to the purpose for which it was used, e.g., documentation was more frequently used for development purposes

Table 1: Summary of experiments features

Ref.	Goal	Participants	Independent variable	Dependent variables	Tasks	Results
[5]	Assessing whether the UML documentation provided helped to reduce the effort required to change the source-code of a software system.	20 + 78 undergraduate (3rd and 4th year) students	Using (or not) UML documentation	Time taken to perform changes, excluding and including diagram modifications, correctness of these changes and quality of the changed design.	Performing several operations related to the systems used.	UML documentation does not provide an advantage as regards time, although it helps to improve the correctness and quality achieved when solving the most complex tasks.
[15]	Assessing whether providing UML documentation reduced the effort required to correctly implement a set of change tasks, and increased the functional correctness and design quality of those changes.	20 professional developers	Using (or not) UML documentation	Time taken to perform changes, excluding and including diagram modifications, correctness of these changes (in terms of the number of submissions of a solution with a fault and their relation to the existing functionalities) and quality of the changed design.	Modifying the existing classes or adding new ones to change the system's behavior (adding new functionalities, changing the original ones, etc.).	UML participants spent more time than the others updating documentation, but using UML was beneficial in terms of functional correctness and helped participants to understand the systems better (especially the more complex ones).
[45]	Relating the level of alignment between UML documents and code and maintainers' efficiency.	21 undergraduate students	Documentation alignment (values more and less)	Efficiency as regards performing maintenance tasks.	Execution of four maintenance tasks on a system.	A more aligned documentation is more helpful during maintenance tasks.
[22]	Investigating whether the origin of UML diagrams influences the maintenance of the corresponding source-code.	20+51+78 MSc students	Origin of the diagrams (forward designed or reverse engineered)	Source-code maintainability, measured by its effectiveness and efficiency.	Five different tasks including adaptive and corrective maintenance tasks.	Participants with a higher ability achieved better scores using the diagrams with a forward design origin, possibly because UML helps more experienced users.
[20]	Analyzing the level of detail in UML diagrams and checking its impact on the maintainability of source-code during a model-centric development.	65 BSc + 16 MSc students	Level of detail of the UML diagrams (high or low)	Modifiability and understandability of the source-code, both measured by their effectiveness and efficiency.	Answering 3 multiple choice questions (understandability) and performing 3 perfective maintenance tasks (modifiability).	Apparently, diagrams with a high level of detail helped participants to understand the system better while those with a low level of detail helped to maintain it. Nevertheless, these results were fairly inconsistent and had no clear tendency.

than for maintenance purposes. Moreover, documentation the up-to-date-ness, accuracy and completeness of document artifacts were identified as the most important and relevant quality attributes as regards improving documentation efficiency. Finally, all these results were used in a set of action-research cycles for a continuous improvement of the company's documentation efficiency.

Finally, Fernández-Sáez et al. [18] present the findings of a survey on the use of UML in software maintenance. A total of 178 professionals from 12 different countries took part in this survey. The main objectives of this survey can be summarized as follows: (i) to explore the extent to which UML diagrams are actually used in industry (59% of the answers indicated the use of a graphical notation, 43% UML), (ii) to acknowledge which was the most effective UML diagram for software maintenance (as expected, class, use case, sequence and activity diagrams), (iii) to discover what the perceived benefits of using UML were (less time needed for a better understanding and, thus, an improved defect detection), and (iv) to contextualize what kind of companies used UML documentation during software maintenance (larger teams seem to use UML more frequently).

2.2 Model-based traceability

Traceability can be considered as an important related concept, since the comprehension of source code supported by the use of models is affected by the possibility of implicitly identifying relationships between source code and software models. When traceability information is explicitly documented in addition to the models, it can help developers to comprehend source code. In the following, we highlight empirical studies dealing with model-based traceability when UML is used to represent artifacts, along with those that analyze source code comprehension and analysis. For example, Lehnert et al. [44] combine impact analysis, multi-perspective modeling, and horizontal traceability analysis to support the specification of models and the development of source code and test cases. They propose a unified meta-model approach that can be supplied by the Eclipse Modeling Framework [16] and a centralized model repository. The approach makes it possible to analyze the dependencies between software artifacts according to the type of change which is applied to them. The idea is to verify the interplay of change operations and dependency relations between models and code with the aim of identifying the propagation of further changes.

Hammad et al. [31] propose an approach with which to automatically determine whether a change in the source code affects the design of the system (i.e., UML class diagrams). The aim of this approach is to maintain consistency between developed code and models by exploiting code-to-design traceability when the source code evolves. The proposed approach, along with the prototype implemented, were assessed by performing a case study based on the commits extracted from four open source projects during a three years period. The results revealed that most of the code changes do not impact on the software models. Furthermore, these commits regard a smaller number of changed files and less lines of code with regard to commits that impact on software models. Another interesting result is that most bug fixes do not impact design.

Settimi et al. [58] assess the effectiveness of information retrieval techniques as regards tracing new and changed requirements to UML artifacts, code, and test cases. The authors summarize the most important result from their research as follows: tracing to UML elements provides a higher perspective of the proposed change than would be possible if links were generated directly to the code and supports the growing trend toward Model Driven

Development. One possible implication is that this kind of link might reduce the effort required to analyze the impact of the changes.

Cariou et al. [12] present an approach that focuses on the object collaboration. This is recognized as an important building block with which to structure object-oriented design in a distributed context. In an attempt to deal with the problem of the deterioration of the collaboration information during the detailed design process, a process and an architecture is proposed to preserve object collaboration information, from the analysis to design and implementation. The idea is to employ UML collaboration diagrams, with the addition of OCL constraints that follows specific rules to suit component specification requirements. This specification can be successively transformed into various low-level implementation designs depending on non-functional constraints by means of a refinement process.

Pavalkis et al. [48] extend UML by defining a model-driven domain specific language engine in order to manage traceability schemas and traceability analysis means. The authors specifically propose a framework that can be used to derive properties in order to trace project artifacts. The authors run several case studies to show how the framework can be used to adapt their solution to a particular development method and domain specific language in a development process, and to automate the maintenance of traceability relations.

The proposal by Tang et al. [61] is focused on the understanding of design rationale to support the detection of inconsistencies, omissions, and conflicts in an architecture design. The model incorporates design rationale, design objects and their relationships, and traceability methods are applied so as to change impact analysis and root cause analysis. The UML notation is used to represent the AREL model, which is an acyclic graph that relates elements of the architecture to their rationale by exploiting the ARtrace directional link (namely, UML stereotyped association) [62].

Pavalkis et al. [49] propose an approach with which to improve vertical traceability of UML models, thus eliminating the additional complexity involved in defining and maintaining traceability information in the software projects. Indeed, traceability information is not statically managed and memorized, but the use of derived properties allows the dynamic calculation of this information, which is then analyzed using dedicated and already existing tool-specific means. The application of the approach to a particular development process has shown that it allows the completeness of the project to be validated and the impact of changes to be analyzed, without boring the users with issues related to the management of traceability information.

3 Background

As the number of empirical studies grows, the need to aggregate evidence from multiple primary empirical studies (e.g., experiments) increases [63]. There are two main reasons for aggregating evidence. Firstly, new research should always take existing knowledge into consideration as its starting point. That is, reviews summarizing the outcomes of various intervention trials are an efficient method by which to obtain the "bottom line" regarding what works and what does not. Secondly, primary empirical studies may together provide answers to research questions, when these studies alone are not sufficient to answer these research questions. In Section 3.1, we first briefly introduce strategies that can be used to summarize and synthesize outcomes from different empirical studies/experiments. When primary studies are synthesized using statistical methods (i.e., using a meta-analysis), it is crucial to verify whether or not these studies are heterogeneous [50]. In Section 3.2 and

Section 3.3, we present some background on how to deal with heterogeneity in meta-analysis studies. Finally, we present the process we have defined to deal with heterogeneity.

3.1 Aggregating Results from Primary Studies

The collection, synthesis, and review of empirical evidence must comply with scientific standards. There are several strategies with which to summarize and synthesize outcomes from different empirical studies/experiments. For example, a systematic literature review is a means used to collect and synthesize empirical evidence from different empirical studies [42]. A systematic literature review is referred to as a secondary study, while the empirical studies in such a review are referred to as primary studies. A systematic literature review has a research question, similar to that in primary studies. If the research question is more general (or if the field of research is less explored) a mapping study may be carried out.

When a set of empirical studies on a topic is collected, synthesis or aggregation takes place. A synthesis based on statistical methods is referred to as a meta-analysis [63]. It can be applied to analyze the outcomes of several dependent and/or independent studies/experiments. The most important advantage of using a meta-analysis is that this kind of secondary study makes it possible to achieve a higher statistical power for the variable of interest than primary studies. Although there is no accepted minimum number of primary studies in a meta-analysis, a minimum of 10 primary studies can be considered acceptable [50].

3.2 Assessing Heterogeneity

Studies may vary. In fact, the assumption that the studies are all representative samples of the overall true effect and only differ owing to sampling error is not always valid. In this situation, the studies are said to be heterogeneous [50]. It is useful to distinguish between different types of heterogeneity. According to Higgins and Green [32], we can distinguish the following kinds of heterogeneity: clinical, methodological, and statistical. Variability in the participants, interventions, and outcomes studied may be described as clinical heterogeneity. Variability in study design and risk of bias may be described as methodological heterogeneity. Finally, variability in the intervention effects being evaluated in the different studies is known as statistical heterogeneity. This kind of heterogeneity can be considered a consequence of clinical heterogeneity or methodological heterogeneity, or both, among the primary studies. In the following part of this section we will focus on statistical heterogeneity and we refer to it simply as heterogeneity.

In a meta-analysis, the means usually employed to assess whether a set of single studies is the Cochran's Q test [50]. This test measures the deviation of observed effect sizes from an underlying overall effect size. The most frequently used cut-off point is 0.1. If the value is lower than this threshold, we can reject the null hypothesis (i.e., the primary studies are not heterogeneous) and we can then assume that studies are heterogeneous. The Cochran's Q test informs us only about the presence of heterogeneity, but it does not report on the extent of that heterogeneity [33]. Possible measures of heterogeneity are I-squared and tau-squared. The I-squared measure is the percentage of total variation across experiments that is owing to heterogeneity rather than chance. Thresholds for the interpretation of I-squared values can be misleading because the importance of inconsistency depends on two main factors: (i) magnitude and direction of effects and (ii) strength of evidence for heterogeneity (e.g., p-value from the chi-squared test, or a confidence interval for the I-squared measure). A guide to the interpretation of I-squared values is based on the intervals suggested by Higgins and Green [32]:

- 0% to 40%: heterogeneity might not be important;
- 30% to 60%: a moderate heterogeneity may be present among the primary studies;
- 50% to 90%: a substantial heterogeneity may be present among the primary studies;
- 75% to 100%: considerable heterogeneity may be present among the primary studies.

Tau-squared is an absolute measure of heterogeneity. It is a measure of the standard deviation of effect sizes across the experiments. Values greater than 1 indicate that primary studies are heterogeneous [50].

Huedo-Medina et al. [33] stated that I squared should be used as a complement to the Cochran's Q test. Since both Tau-squared and I-squared are measures of heterogeneity, these measures can be considered both as a complement to the Cochran's Q test. That is, the heterogeneity of the primary studies can be measured by either (or both) Tau-squared or I-squared if the Cochran's Q test rejects the null hypothesis that these studies are not heterogeneous.

If studies are not heterogeneous (i.e., they are homogeneous), they should be combined in a meta-analysis using a fixed effect model. This model assumes that the size of the treatment effect is the same (fixed) across all the experiments.

3.3 Dealing with Heterogeneity

A number of options are available if (statistical) heterogeneity is identified among a group of primary studies that would otherwise be considered suitable for a meta-analysis. For example, Higgins and Green [32] suggested:

1. *Check again that the data are correct*. Severe heterogeneity can indicate that data have been incorrectly extracted and/or used. For example, if standard errors have mistakenly been entered as standard deviations for continuous outcomes [32].
2. *Do not do a meta-analysis*. If there is considerable variation in results, and particularly if there is inconsistency in the direction of the effect, it may be misleading to quote an average value for the effect.
3. *Explore heterogeneity*. The goal is to determine the causes of heterogeneity. This is problematic since there are often many characteristics that vary across primary studies from which one may choose. Heterogeneity may be explored by conducting sub-group analyses. Each of these groups should have a minimum of 4 studies/experiments [24]. Explorations of heterogeneity can at best lead to the generation of hypotheses. They should be interpreted with caution. Investigations of heterogeneity when there are very few studies are of questionable value.
4. *Ignore heterogeneity*. Heterogeneity can be ignored and a fixed effect meta-analyses can be performed. The pooled effect estimate from a fixed effect meta-analysis is normally interpreted as being the best estimate of the intervention effect. However, the existence of heterogeneity suggests that there may not be a single intervention effect but rather a distribution of intervention effects. The pooled fixed effect estimate may be thus an intervention effect that does not actually exist in any population, and, therefore, have a confidence interval that is both meaningless and too narrow.
5. *Perform a random effects meta-analysis*. A random effects meta-analysis may be used to incorporate heterogeneity among primary studies. This is not a substitute for a thorough

investigation of heterogeneity. It should be intended primarily for heterogeneity that cannot be explained.

6. *Change the effect measure*. Heterogeneity may be an artificial consequence of an inappropriate choice of the effect measure (dependent variable). Furthermore, the choice of the effect measure for dichotomous outcomes (odds ratio, relative risk, or risk difference) may affect the degree of heterogeneity among results. When control group risks vary, homogeneous odds ratios (or risk ratios) could lead to heterogeneous risk differences (and vice versa).
7. *Exclude studies*. Heterogeneity may be owing to the presence of one or two outlying studies. That is, the results of these studies could conflict with the results of the remaining studies. In general, it is unwise to exclude studies on the basis of their results. This may introduce bias into the meta-analysis results. However, if an obvious reason for the outlying result is apparent, the study might be removed with more confidence. It is advisable to perform analyses both with and without outlying studies as part of a sensitivity analysis. Whenever possible, potential sources of diversity that might lead to heterogeneity should be specified in the experimental protocol. When excluding studies, a researcher should also take into consideration the number of studies to be then considered in the meta-analysis, thus avoiding losing the representativeness of results.

3.4 A Process Based Approach to Deal with Heterogeneity

We employed the strategies by Higgins and Green [32] as a basis to define a process with which to better deal with (devised statistical) heterogeneity in meta-analysis studies. Figure 1 shows this process as an activity diagram with object flow, where the activities are the phases of the process and the objects represent the input/output to these phases. In particular, the process first suggests checking data from individual experiments to be sure that they are correct (*CheckingData*). This is performed before checking that the secondary studies are heterogenous. The phasse in charge of assessing whether secondary studies are statistically heterogenous is *AssessingHeterogeneity*. If the secondary studies are not heterogenous, a meta-analysis will take place in *PerformingFixedEffectModel*. If the secondary studies are heterogeneous, the researcher can decide not carry out a meta-analysis (the flow is placed in the end-node) or to go ahead with the process. In the latter case, the researcher can decide whether or not to ignore heterogeneity. Ignoring heterogeneity implies the execution of *PerformingFixedEffectModel*. On the other hand, the researcher can decide whether or not to incorporate the heterogeneity into a meta-analysis. If the researcher decides to incorporate the heterogeneity, the meta-analysis will take place by executing *PerformingRandom-EffectsModel*. That is, a random effects model will be applied to all the primary studies. It is also possible to either explore or not explore heterogeneity. If heterogeneity is not explored, the researcher can decide to choose a different effect measure (or dependent variable) in order to aggregate the results in a meta-analysis (*ChangingEffectMeasure*). There are two ways in which to carry out an exploring: by excluding one or two outlying primary studies (*ExcludingStudies*) or by identifying sub-groups of experiments (*IdentifyingSubGroups*). In the case of the researcher excluding one or two outlying studies, meta-analysis takes place with a single group of studies. This implies that the process in Figure 1 is instantiated only once. If sub-groups of experiments are identified, the process is instantiated for each of these sub-groups.

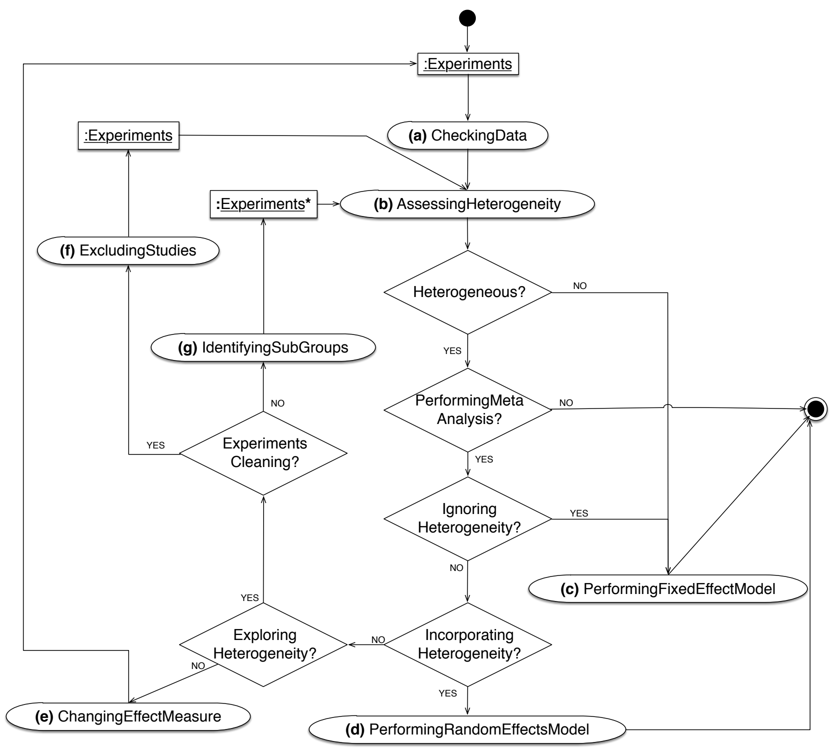


Fig. 1: The process defined to deal with statistical heterogeneity, in which each phase has a label (i.e., a, b, c, d, e, f, or g) associated with it. This allows a rapid reference to the phases in the process.

In order to show the different instances[1] of the process shown in Figure 1, we used regular expressions in which the symbols are those used to label the phases of the process shown in Figure 1: a, b, c, d, e, f, and g. For example, the label for *CheckingData* is a. We also considered the empty symbol (i.e., ϵ). The regular expression defined is shown as follows:

$$ab((eab)^*(gb)^*(fb)^*)^*(c|d|\epsilon)$$

We used regular expressions to provide a more compact representation of the instances of this process. In particular, our solution makes it possible to establish a clear link between the instances in our process and the sentences in the language described by means of the regular expression shown previously. We shall also use the sentences from the regular expression to facilitate our discussion on how we dealt with heterogeneity in the study presented in this paper. For example, the sentence *abc* means that we checked data, assessed heterogeneity, and performed a fixed effect model to aggregate results. However, our compact representation does not make it evident whether we execute *PerformingFixed EffectModel* because heterogeneity is ignored or because the experiments are homogeneous. We deal with this ambiguity by underlining sentences in the regular expression defined to indicate that the

[1] An instance of a process is a sequence of the activities/phases.

experiments are homogeneous. The sentence *abc*, therefore, indicates that the experiments are heterogeneous, while *abc* indicates that they are homogeneous. In both cases the sequences in which the phases are performed is: *CheckingData*, *AssessingHeterogeneity*, and *PerformingFixed EffectModel*.

4 Our Long-Term Investigation

In a survey we conducted in 2009 [56], the main results suggested that, in order to deal with software maintenance and evolution tasks, many of the companies interviewed use UML diagrams produced in the requirements analysis[2] phase and, in a few cases, those produced in the design[3] phase. The most frequently used UML diagrams were: use case, class, and sequence diagrams. Another result of this survey was: maintenance operations were performed by practitioners with a few years of experience in software development and maintenance and maintenance operations were supported by models produced in the analysis and design phases. To this end, companies generally employ developers with a Bachelor's or Master's degree in Computer Science and with between 1 and 5 years' experience.

The main results of the aforementioned industrial survey were used as a basis to begin the long-term research shown in this paper. We quantitatively studied to what extend developers understand source code when they were provided with source code alone or with source code and UML software models together. Our research consisted of the following two main directions, which were carried out in parallel:

– In the first direction of our long term research, we studied the comprehension of source code when it was complemented with analysis models (i.e., models produced in the requirements engineering process) based on the UML notation: use case diagrams, class diagrams, and sequence diagrams. In particular, use case diagrams and use cases (textual description of the use cases in the use case diagrams) were employed to represent functional requirements. These requirements are represented with successful use cases. The participants were also given exceptional and/or boundary conditions. Class diagrams were used to abstract the objects from the problem domain (i.e., the object or conceptual model), while sequence diagrams (also produced in the requirements engineering phase) were used to model the dynamic and/or functional behavior of the software [10]. First, we conducted a pilot study with Computer Science Bachelor's degree students at the University of Basilicata. The results of this pilot study were presented in [28] and can be summarized as follows: the use of analysis models does not significantly improve the comprehension of source-code. We successively conducted a family of four controlled experiments on this subject [53], the goal of which was to strengthen the findings obtained in the pilot study. The experiments were carried out with students and practitioners from Italy and Spain who had different abilities and levels of experience with the UML. The results attained indicated that UML analysis models did not appear to improve source-code comprehensibility, thus confirming the results from the pilot study.
– In the second direction, we conducted two kinds of controlled experiments. The main goal of the second direction of our research was to study the comprehension of source

[2] It is also called requirements engineering process and it is the process of determining user expectations (i.e., requirements) as regards a new or modified product.

[3] It maps the requirements onto the software architecture that defines the components, their interfaces and behaviors. The design document describes a plan with which to implement the requirements.

code when it was complemented with design models (i.e., models produced in the design phase) based on the UML. In particular, the two kinds of controlled experiments were conducted in parallel and pursued the following main goals:

(i) Assessing the potential benefits derived from the use of UML class and sequence diagrams (both produced in the design phase) as regards the comprehension of object-oriented source-code [27]. Two experiments with Computer Science Bachelor's and Master's degree students were, therefore, conducted. The data analysis revealed that those participants with more experience of the UML and computer programming (i.e., Master's degree students) benefitted from the use of UML models produced in the design phase (from here on UML design models, also).

(ii) Investigating whether providing source code with UML class diagrams used to graphically document design-pattern instances[4] improves source-code comprehensibility. This implies that more diagrams refer to the same piece of software and each of them can be seen as an excerpt of the entire class diagram of that piece of software. This is the main difference between this aspect of our second research direction and that just before. We conducted an experiment with Master's degree students [54]. The control group of this experiment comprised students who were given source code alone without any reference to the design-pattern instances contained in it. We carried out four successive controlled experiments with participants who had different experience as regards programming and software modeling (i.e., Bachelor's, Master's, and PhD students and practitioners) [55]. The effect of textually documented design-pattern instances was also studied. Data regarding this kind of documentation was clearly not considered in the study presented in this paper, since the UML was not used.

All 12 experiments (primary studies) briefly described above and summarized in Table 2 were carried out by following the recommendations provided in [40,43,63]. In this section, we summarize the planning and the operation phases of these experiments and provide the most salient information concerning the study presented in this paper.

The experiments are reported according to the guidelines suggested by Jedlitschka et al. [38]. For replication purposes, we have made the raw data regarding of all our experiments available on the web.[5]

4.1 Goal

According to the Goal Question Metrics template [8], the goal of the study presented in this paper is: *to analyze* the use of UML analysis and design models *for the purpose of* understanding their utility *with respect to* the comprehensibility of object-oriented source code *from the point of view of* the software engineer *in the context of* students and practitioners.

4.2 Context Selection

We used different software systems and UML diagrams in our study. The systems used were those described in the fifth column of Table 2, while the diagrams are those shown in the

[4] Design-pattern instances can be seen as a micro-architecture that developers copy and adapt to their particular designs in order to solve the recurrent problem described by the design pattern [10,25].

[5] www2.unibas.it/gscanniello/SourceCodeComprMetaAnalysis/data.xlsx

second column of this table. All the experimental objects were desktop applications and were implemented in Java. We used the Music Shop[6] and Theater Ticket Reservation[7] applications. Moreover, their models were created in a course on Advanced Object-Oriented Programming (AOOP). The lecturer of this course was involved in neither the study presented here nor those shown in Table 2, which allowed us to mitigate possible threats to construct validity (Experimenters' Expectancies). We used the source code that the lecturer selected from among the software systems developed by the students on the AOOP course. We did not have any control over the selection process. However, we reviewed the documentation and models to find possible issues. It was not necessary make any considerable modifications. We only removed possible typographical errors and indented source code when appropriate. Source-code comments were removed to avoid their presence having any effect on the results. That is, the effect of source-code comments could have been confused (or could have interacted) with the main factor being studied. This design choice could have affected external validity and it is further discussed in Section 5.4.2. The students that developed the experimental objects did not participate in the experiments. In order to deal with the threat to external validity, we also used open-source software. We selected a chunk (i.e., a vertical slice) of JHotDraw v5.1. This chunk included: (i) 10 instances of design patterns in total — two instances for the State design pattern and one instance for the following design patterns: Adapter, Strategy, Decorator, Composite, Observer, Command, Template Method, and Prototype — and (ii) the design patterns considered were well-known and widely adopted [25]. We documented the design-pattern instances present in the source code using both the JHotDraw documentation and the PMARt dataset [30]. This allowed us to document both intentional and unintentional design-pattern instances. Although traceability links are important as regards identifying relationships between source code and software models (see Section 2.2), we avoided providing them to the participants. The rationale for this was that making this information available to a developer could affect source code comprehensibility in an undesirable way, i.e., concealing the effect of the use of software models on source code comprehension. However, this design choice could have affected the external validity, since traceability links could be available (e.g., post-facto) to support program comprehension tasks in the software industry [4].

We conducted all the experiments, with the exception of DePra (see Table 2), in research laboratories. DePra was conducted at the participants' companies. All the experiments were conducted under controlled conditions. The most participants' salient characteristics are summarized in Table 2 (third column). Participation in the experiments was on a voluntary basis. The participants were not paid. Each participant took part in only one experiment. Further information on the experimental objects and their selection process, along with the characteristics of the participants in the experiments, can be found in [27,28,53–55].

[6] This is a software system that is used to sell and manage CDs/DVDs in a music shop. The feature *search for a singer* was used in the experiments: the user inserts a string (e.g., the surname of the singer), and the system then searches for all the singers that satisfy the search criterion and shows them in a list of the associated information.

[7] IThis is a software system with which to book and buy theater tickets. The feature *buy a theater ticket* was used in the experiments: the system shows the list of the available tickets for a given theater and performance, and the user then chooses the ticket and inserts data about the spectator.

Table 2: Summary of the experiments[*]

Experiment	UML Diagrams	Number of Participants and Kind	Design	Exp. Objects	Results	
					Comprehension	Completion time
AnBsc [28]	Use case, class, and sequence	16 3rd year Bachelor's degree students	- One-factor-with more treatments - Randomized	A chunk of Music Shop software system implemented in Java	The difference is not statistically significant	Not considered
DeBscExp1 [27]	Class and sequence	16 2nd year Bachelor's degree students	- Crossover design - Randomized	A chunk of a Theater Ticket Reservation system implemented in Java	The difference is not statistically significant	The difference is statistically significant. More time taken when models used
DeMscExp1 [27]		16 second year Master's degree students			The difference is statistically significant. Better comprehension when models used	The difference is not statistically significant
AnMscExp1 [53]	Use case, class, and sequence	24 1st year Master's degree students	- Crossover design - Ability as blocking factor		The difference is not statistically significant	Not considered
AnMscExp2 [53]		22 2nd year Master's degree students			The difference is not statistically significant	
AnMscExp3 [53]		22 1st year Master's degree students			The difference is statistically significant. Better comprehension when models not used	
AnPra [53]		18 Practitioners			The difference is not statistically significant	
DeMscExp2 [54]	Class	24 1st year Master's degree students	-One-factor-with more treatments - Ability as blocking factor for students - Years of working experience as blocking factor for practitioners	JHotDraw: a two-dimensional graphics framework for structured drawing editors implemented in Java. A chunk (i.e., vertical slice) of the entire software was selected	Analysis of comprehension not presented	The difference is statistically significant. Less time taken when models used
DePra [55]		16 Practitioners			The difference is statistically significant. Better comprehension when models used	The difference is not statistically significant
DeMscExp3 [55]		16 1st year Master's degree students			The difference is not statistically significant	
DeBscExp2 [55]		15 3rd year Bachelor's degree students				
DePhd [55]		10 Ph.D students			The difference is statistically significant. Better comprehension when models used	

[*] The labels of the experiments are expressed using the upper-camel case notation. Each compound word has a specific meaning. The first word suggests the kind of UML-based documentation: (An) stands for those documents produced in the analysis phase and (De) stands for those documents produced in the design phase. The second word indicates the kind of participant. For example, Pra and Msc stand for practitioner and Master's degree student, respectively. The third word is a progressive id for the experiment. We used id to better discern each experiment that was based on the same kind of documentation and involved the same kind of participants.

4.3 Selection of Variables

In each experiment, we considered those participants who were given source code alone as comprising the *control group*, while the *treatment group* comprised students who were given source code with software models based on the UML. The independent variable (from here on manipulated factor or main factor, also) considered in each primary study was, therefore, $method$. This variable is nominal and can assume the following two values: *models* (UML-based models and source code without comments) and *source code* (source code without comments).

The effect of the manipulated factor was analyzed on the following chosen constructs:

- **Comprehension**. This denotes the comprehension level of the source code achieved by a software engineer.
- **Completion time**. This denotes the time a software engineer takes to accomplish a comprehension task.

We used questionnaires to assess source-code comprehensibility. The correctness and completeness of the answers provided in these questionnaires were quantitatively evaluated by using an information-retrieval-based approach. Each answer was provided in the form of string items (e.g., a sequence of method/class names and/or the text messages shown to a user), which were compared with the expected items. We measured the correctness of the answers by using the precision measure, while the completeness was measured by means of the recall measure. In order to obtain a single measure for all the questions in a questionnaire, we computed the overall average of the F-measure values (i.e., a balanced harmonic mean of precision and recall). The F-measure was used to estimate the comprehension variable. F-measure values range in between 0 and 1. This was the dependent variable (or response variable) used to assess the comprehension construct. The higher the value of this variable, the greater the comprehensibility of source-code was.

We estimated the completion time construct using the overall time (expressed in minutes) taken to answer a comprehension questionnaire. The higher the value of time, the greater the effort[8] required to accomplish a comprehension task.

4.4 Design

We used different kinds of designs in the experiments. As shown in Table 2, we used crossover designs in DeMscExp1, AnMscExp1, AnMscExp2, AnMscExp3, and AnPra. In the remaining experiments, we adopted the one-factor-with two treatments design [63]. This kind of experimental does not suffer from the presence of a possible carry-over effect,[9] while the crossover design may do so. It is worth noting that the design of some experiments was randomized, while in others we used the participants' ability as a blocking factor (see Table 2 for details). When applicable, randomization allowed us to mitigate carry-over that we had already analyzed in the primary studies. In all the experiments, the participants accomplished the task alone, that is, they did not work in a group to accomplish a comprehension task.

[8] The time was an approximation of comprehension effort. This complies with the ISO/IEC 9126 standard [35] (and subsequent versions), in which effort is the productive time associated with a specific project task.

[9] If a participant is tested first under the experimental condition A and then under the experimental condition B, she/he could potentially perform better or worse under condition B.

4.5 Experimental Tasks and Operation

All the participants were asked to fill in a comprehension questionnaire and a post-experiment survey questionnaire. The composition of both these questionnaires depended on the experiment and the tasks. We formulated the questions in the comprehension questionnaires using a similar form/schema. In addition, these questions were formulated to assess comprehension of the source code that we believed to be more relevant and concerned understanding concepts in this source code, which (as suggested by Sillitto et al. [60]) involved multiple relationships and software entities. Further details on the experimental tasks and the experimental procedure can be found in [27,28,53–55].

4.6 Analysis Procedure

Th results of a meta-analysis are commonly displayed graphically as "forest plots" [51]. This kind of pictorial representation provides a quick and easy means to illustrate the relative strength of treatment effects. Forest plots display point estimates and confidence intervals for individual experiments, in addition to an estimate of the overall summary effect size. This notation also shows the extent to which each experiment contributes to the overall result.

5 Results and Discussion

In Table 3, we show some sentences from the language defined by employing the regular expression reported in Section 3.4. For example, ab covers the case of no heterogeneity in which a meta-analysis is not executed, while abc and abd represent the case in which heterogeneity is ignored or incorporated and a meta-analysis is executed, respectively. Conversely, in order to cover the case of exploring heterogeneity, we performed sub-group analyses for the dependent variables Comprehension and Completion time, since the experiments were heterogeneous. In particular, we grouped the experiments according to the kind of models used, namely An (i.e., UML-based models produced in the requirements engineering process) and De (UML-based models produced in the design phase). These cases are covered by the sentences $ab(gb)^*c$ and $ab(gb)^*d$.

Furthermore, $ab(fb)^*$, $ab(fb)^*c$, and $ab(fb)^*d$ cover the cases of the application of the process shown in Figure 1 when we cleaned the experiment set by excluding those studies that involved participants with little experience, namely AnBsc, DeBscExp1, and DeBsc-Exp2. We also considered the cases of not exploring the heterogeneity and employing a further dependent variable, namely Efficiency,[10] which are covered by the sentences $ab(eab)^*$, $ab(eab)^*c$, and $ab(eab)^*d$.

In the following subsections, we first report the results obtained and then discuss them according to the cases shown in Table 3. We then present the implications of our study and conclude with a discussion regarding the threats to validity.

[10] Efficiency is a derived measure that is computed as the ratio between comprehension and completion time. Task efficiency is a ratio measure and estimates a participant's efficiency as regards the execution of a comprehension task. The larger the efficiency value, the better it is. The perspective we adopted is that of quality in use (e.g., [36,37]), since efficiency measures source code comprehension achieved during the expenditure of available models.

Table 3: Instances of the process shown in Figure 1 considered

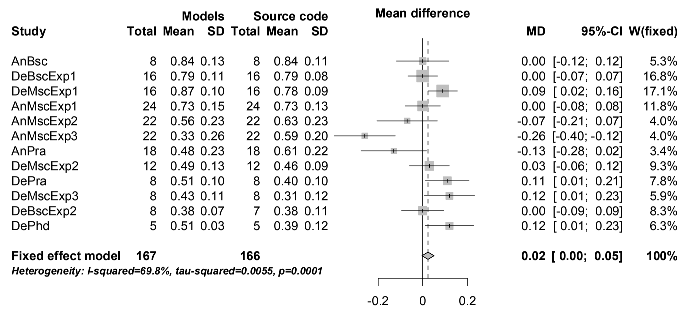
Sentence	Dependent Variable
ab	Comprehension, Completion Time
abc	Comprehension, Completion Time
abd	Comprehension, Completion Time
$abgbc$	Comprehension, Completion Time
$abgbd$	Completion Time
$abfb$	Comprehension, Completion Time
$abfbc$	Comprehension, Completion Time
$abfbd$	Comprehension, Completion Time
$abeab$	Efficiency
$abeabc$	Efficiency
$abeabd$	Efficiency

5.1 Meta-Analysis Results

We summarize the results of each experiment, by employing the descriptive statistics of the measures of the dependent variables. The descriptive statistics for Comprehension (i.e., mean, standard deviation, and number of observations) grouped by method are shown in the forest plot in Figure 2 (for each of the cases considered shown in Table 3). The same descriptive statistics for Completion time and Efficiency are reported in Figure 3 and Figure 4, respectively. It is worth mentioning that we do not show any results for the sentences ab, $abfb$, $abgb$, and $abeab$ (see Table 3), because the process in Figure 1 does not require the execution of a meta-analysis. Some other sentences are also not reported (e.g., those that excessively reduce the number of experiments in the analyses, see Section 3).

The results are synthesized by means of the Mean Differences (MDs) of the outcome measures of the experiments. This is possible because the experiments have the same outcome measures for each dependent variable studied.

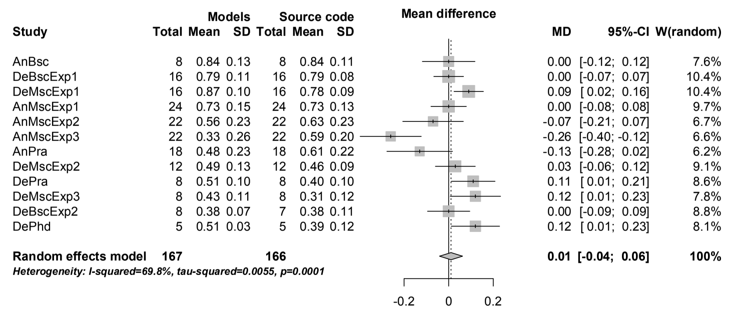
Results concerning the testing of heterogeneity are also shown at the bottom of each forest plot (see, for example, the left-hand side of Figure 2(a)). With regard to Comprehension, the results of the Cochran's Q test (see Figure 2(a) — sentence abc) suggest that the experiments were heterogeneous (p=0.0001) while the I-squared values indicates a substantial/considerable heterogeneity that is also confirmed by the Tau-squared value. We can, therefore, incorporate heterogeneity and apply a random effects model (see Figure 2(b) — sentence abd).



Study	Models Total	Models Mean	Models SD	Source code Total	Source code Mean	Source code SD	MD	95%-CI	W(fixed)
AnBsc	8	0.84	0.13	8	0.84	0.11	0.00	[-0.12; 0.12]	5.3%
DeBscExp1	16	0.79	0.11	16	0.79	0.08	0.00	[-0.07; 0.07]	16.8%
DeMscExp1	16	0.87	0.10	16	0.78	0.09	0.09	[0.02; 0.16]	17.1%
AnMscExp1	24	0.73	0.15	24	0.73	0.13	0.00	[-0.08; 0.08]	11.8%
AnMscExp2	22	0.56	0.23	22	0.63	0.23	-0.07	[-0.21; 0.07]	4.0%
AnMscExp3	22	0.33	0.26	22	0.59	0.20	-0.26	[-0.40; -0.12]	4.0%
AnPra	18	0.48	0.23	18	0.61	0.22	-0.13	[-0.28; 0.02]	3.4%
DeMscExp2	12	0.49	0.13	12	0.46	0.09	0.03	[-0.06; 0.12]	9.3%
DePra	8	0.51	0.10	8	0.40	0.10	0.11	[0.01; 0.21]	7.8%
DeMscExp3	8	0.43	0.11	8	0.31	0.12	0.12	[0.01; 0.23]	5.9%
DeBscExp2	8	0.38	0.07	7	0.38	0.11	0.00	[-0.09; 0.09]	8.3%
DePhd	5	0.51	0.03	5	0.39	0.12	0.12	[0.01; 0.23]	6.3%
Fixed effect model	**167**			**166**			**0.02**	**[0.00; 0.05]**	**100%**

Heterogeneity: I-squared=69.8%, tau-squared=0.0055, p=0.0001

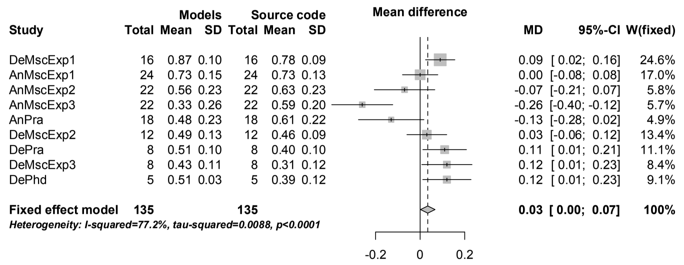
(a) abc - Fixed effect model on all the studies



Study	Models Total	Models Mean	Models SD	Source code Total	Source code Mean	Source code SD	MD	95%-CI	W(random)
AnBsc	8	0.84	0.13	8	0.84	0.11	0.00	[-0.12; 0.12]	7.6%
DeBscExp1	16	0.79	0.11	16	0.79	0.08	0.00	[-0.07; 0.07]	10.4%
DeMscExp1	16	0.87	0.10	16	0.78	0.09	0.09	[0.02; 0.16]	10.4%
AnMscExp1	24	0.73	0.15	24	0.73	0.13	0.00	[-0.08; 0.08]	9.7%
AnMscExp2	22	0.56	0.23	22	0.63	0.23	-0.07	[-0.21; 0.07]	6.7%
AnMscExp3	22	0.33	0.26	22	0.59	0.20	-0.26	[-0.40; -0.12]	6.6%
AnPra	18	0.48	0.23	18	0.61	0.22	-0.13	[-0.28; 0.02]	6.2%
DeMscExp2	12	0.49	0.13	12	0.46	0.09	0.03	[-0.06; 0.12]	9.1%
DePra	8	0.51	0.10	8	0.40	0.10	0.11	[0.01; 0.21]	8.6%
DeMscExp3	8	0.43	0.11	8	0.31	0.12	0.12	[0.01; 0.23]	7.8%
DeBscExp2	8	0.38	0.07	7	0.38	0.11	0.00	[-0.09; 0.09]	8.8%
DePhd	5	0.51	0.03	5	0.39	0.12	0.12	[0.01; 0.23]	8.1%
Random effects model	**167**			**166**			**0.01**	**[-0.04; 0.06]**	**100%**

Heterogeneity: I-squared=69.8%, tau-squared=0.0055, p=0.0001

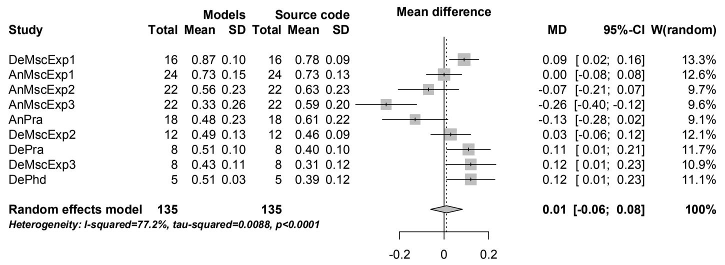
(b) abd - Random effects model on all the studies



Study	Models Total	Models Mean	Models SD	Source code Total	Source code Mean	Source code SD	MD	95%-CI	W(fixed)
DeMscExp1	16	0.87	0.10	16	0.78	0.09	0.09	[0.02; 0.16]	24.6%
AnMscExp1	24	0.73	0.15	24	0.73	0.13	0.00	[-0.08; 0.08]	17.0%
AnMscExp2	22	0.56	0.23	22	0.63	0.23	-0.07	[-0.21; 0.07]	5.8%
AnMscExp3	22	0.33	0.26	22	0.59	0.20	-0.26	[-0.40; -0.12]	5.7%
AnPra	18	0.48	0.23	18	0.61	0.22	-0.13	[-0.28; 0.02]	4.9%
DeMscExp2	12	0.49	0.13	12	0.46	0.09	0.03	[-0.06; 0.12]	13.4%
DePra	8	0.51	0.10	8	0.40	0.10	0.11	[0.01; 0.21]	11.1%
DeMscExp3	8	0.43	0.11	8	0.31	0.12	0.12	[0.01; 0.23]	8.4%
DePhd	5	0.51	0.03	5	0.39	0.12	0.12	[0.01; 0.23]	9.1%
Fixed effect model	**135**			**135**			**0.03**	**[0.00; 0.07]**	**100%**

Heterogeneity: I-squared=77.2%, tau-squared=0.0088, p<0.0001

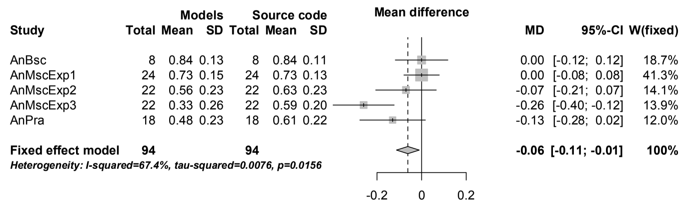
(c) $ab(fb)^{*}c$ - Fixed effect model after excluding studies



Study	Models Total	Models Mean	Models SD	Source code Total	Source code Mean	Source code SD	MD	95%-CI	W(random)
DeMscExp1	16	0.87	0.10	16	0.78	0.09	0.09	[0.02; 0.16]	13.3%
AnMscExp1	24	0.73	0.15	24	0.73	0.13	0.00	[-0.08; 0.08]	12.6%
AnMscExp2	22	0.56	0.23	22	0.63	0.23	-0.07	[-0.21; 0.07]	9.7%
AnMscExp3	22	0.33	0.26	22	0.59	0.20	-0.26	[-0.40; -0.12]	9.6%
AnPra	18	0.48	0.23	18	0.61	0.22	-0.13	[-0.28; 0.02]	9.1%
DeMscExp2	12	0.49	0.13	12	0.46	0.09	0.03	[-0.06; 0.12]	12.1%
DePra	8	0.51	0.10	8	0.40	0.10	0.11	[0.01; 0.21]	11.7%
DeMscExp3	8	0.43	0.11	8	0.31	0.12	0.12	[0.01; 0.23]	10.9%
DePhd	5	0.51	0.03	5	0.39	0.12	0.12	[0.01; 0.23]	11.1%
Random effects model	**135**			**135**			**0.01**	**[-0.06; 0.08]**	**100%**

Heterogeneity: I-squared=77.2%, tau-squared=0.0088, p<0.0001

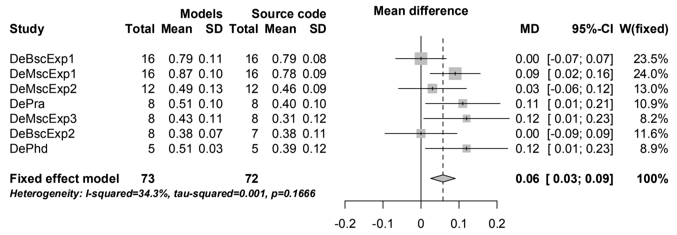
(d) $ab(fb)^{*}d$ - Random effects model after excluding studies



Study	Models Total	Models Mean	Models SD	Source code Total	Source code Mean	Source code SD	MD	95%-CI	W(fixed)
AnBsc	8	0.84	0.13	8	0.84	0.11	0.00	[-0.12; 0.12]	18.7%
AnMscExp1	24	0.73	0.15	24	0.73	0.13	0.00	[-0.08; 0.08]	41.3%
AnMscExp2	22	0.56	0.23	22	0.63	0.23	-0.07	[-0.21; 0.07]	14.1%
AnMscExp3	22	0.33	0.26	22	0.59	0.20	-0.26	[-0.40; -0.12]	13.9%
AnPra	18	0.48	0.23	18	0.61	0.22	-0.13	[-0.28; 0.02]	12.0%
Fixed effect model	**94**			**94**			**-0.06**	**[-0.11; -0.01]**	**100%**

Heterogeneity: I-squared=67.4%, tau-squared=0.0076, p=0.0156

(e) $ab(gb)^{*}c$ - Fixed effect model on the sub-group An

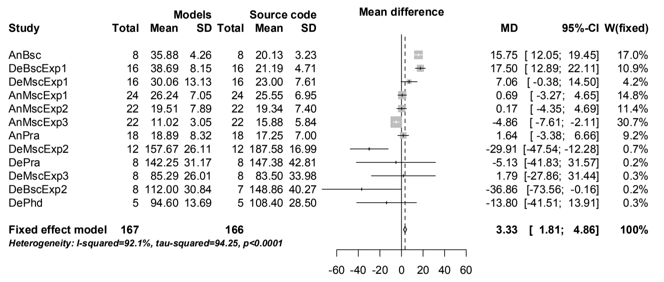


Study	Models Total	Models Mean	Models SD	Source code Total	Source code Mean	Source code SD	MD	95%-CI	W(fixed)
DeBscExp1	16	0.79	0.11	16	0.79	0.08	0.00	[-0.07; 0.07]	23.5%
DeMscExp1	16	0.87	0.10	16	0.78	0.09	0.09	[0.02; 0.16]	24.0%
DeMscExp2	12	0.49	0.13	12	0.46	0.09	0.03	[-0.06; 0.12]	13.0%
DePra	8	0.51	0.10	8	0.40	0.10	0.11	[0.01; 0.21]	10.9%
DeMscExp3	8	0.43	0.11	8	0.31	0.12	0.12	[0.01; 0.23]	8.2%
DeBscExp2	8	0.38	0.07	7	0.38	0.11	0.00	[-0.09; 0.09]	11.6%
DePhd	5	0.51	0.03	5	0.39	0.12	0.12	[0.01; 0.23]	8.9%
Fixed effect model	**73**			**72**			**0.06**	**[0.03; 0.09]**	**100%**

Heterogeneity: I-squared=34.3%, tau-squared=0.001, p=0.1666

(f) $ab(gb)^{*}c$ Fixed effect model on the sub-group De

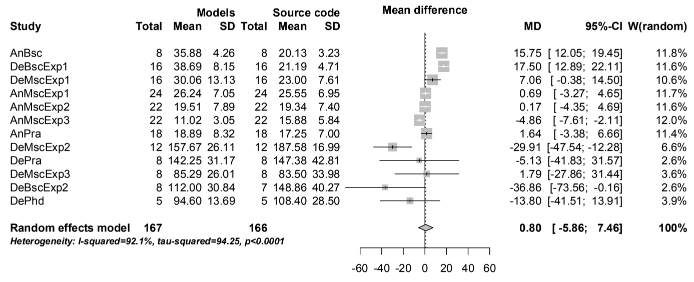
Fig. 2: Forest plots for Comprehension



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(fixed)
AnBsc	8	35.88	4.26	8	20.13	3.23	15.75	[12.05; 19.45]	17.0%
DeBscExp1	16	38.69	8.15	16	21.19	4.71	17.50	[12.89; 22.11]	10.9%
DeMscExp1	16	30.06	13.13	16	23.00	7.61	7.06	[-0.38; 14.50]	4.2%
AnMscExp1	24	26.24	7.05	24	25.55	6.95	0.69	[-3.27; 4.65]	14.8%
AnMscExp2	22	19.51	7.89	22	19.34	7.40	0.17	[-4.35; 4.69]	11.4%
AnMscExp3	22	11.02	3.05	22	15.88	5.84	-4.86	[-7.61; -2.11]	30.7%
AnPra	18	18.89	8.32	18	17.25	7.00	1.64	[-3.38; 6.66]	9.2%
DeMscExp2	12	157.67	26.11	12	187.58	16.99	-29.91	[-47.54; -12.28]	0.7%
DePra	8	142.25	31.17	8	147.38	42.81	-5.13	[-41.83; 31.57]	0.2%
DeMscExp3	8	85.29	26.01	8	83.50	33.98	1.79	[-27.86; 31.44]	0.3%
DeBscExp2	8	112.00	30.84	7	148.86	40.27	-36.86	[-73.56; -0.16]	0.2%
DePhd	5	94.60	13.69	5	108.40	28.50	-13.80	[-41.51; 13.91]	0.3%
Fixed effect model	**167**			**166**			**3.33**	**[1.81; 4.86]**	**100%**

Heterogeneity: I-squared=92.1%, tau-squared=94.25, p<0.0001

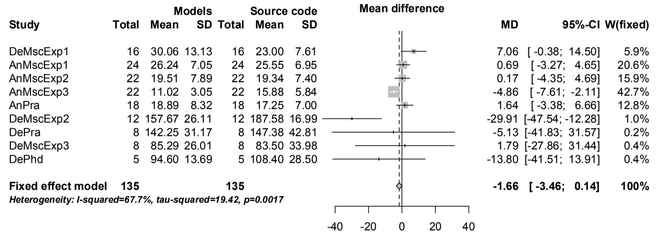
(a) abc - Fixed effect model on all the studies



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(random)
AnBsc	8	35.88	4.26	8	20.13	3.23	15.75	[12.05; 19.45]	11.8%
DeBscExp1	16	38.69	8.15	16	21.19	4.71	17.50	[12.89; 22.11]	11.6%
DeMscExp1	16	30.06	13.13	16	23.00	7.61	7.06	[-0.38; 14.50]	10.6%
AnMscExp1	24	26.24	7.05	24	25.55	6.95	0.69	[-3.27; 4.65]	11.7%
AnMscExp2	22	19.51	7.89	22	19.34	7.40	0.17	[-4.35; 4.69]	11.6%
AnMscExp3	22	11.02	3.05	22	15.88	5.84	-4.86	[-7.61; -2.11]	12.0%
AnPra	18	18.89	8.32	18	17.25	7.00	1.64	[-3.38; 6.66]	11.4%
DeMscExp2	12	157.67	26.11	12	187.58	16.99	-29.91	[-47.54; -12.28]	6.6%
DePra	8	142.25	31.17	8	147.38	42.81	-5.13	[-41.83; 31.57]	2.6%
DeMscExp3	8	85.29	26.01	8	83.50	33.98	1.79	[-27.86; 31.44]	3.6%
DeBscExp2	8	112.00	30.84	7	148.86	40.27	-36.86	[-73.56; -0.16]	2.6%
DePhd	5	94.60	13.69	5	108.40	28.50	-13.80	[-41.51; 13.91]	3.9%
Random effects model	**167**			**166**			**0.80**	**[-5.86; 7.46]**	**100%**

Heterogeneity: I-squared=92.1%, tau-squared=94.25, p<0.0001

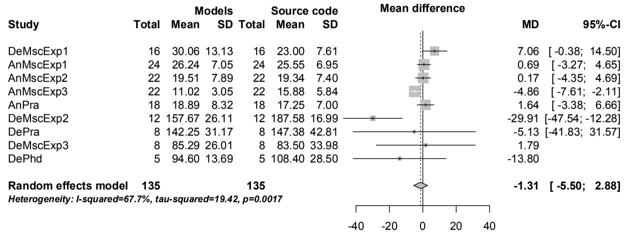
(b) abd - Random effects model on all the studies



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(fixed)
DeMscExp1	16	30.06	13.13	16	23.00	7.61	7.06	[-0.38; 14.50]	5.9%
AnMscExp1	24	26.24	7.05	24	25.55	6.95	0.69	[-3.27; 4.65]	20.6%
AnMscExp2	22	19.51	7.89	22	19.34	7.40	0.17	[-4.35; 4.69]	15.9%
AnMscExp3	22	11.02	3.05	22	15.88	5.84	-4.86	[-7.61; -2.11]	42.7%
AnPra	18	18.89	8.32	18	17.25	7.00	1.64	[-3.38; 6.66]	12.8%
DeMscExp2	12	157.67	26.11	12	187.58	16.99	-29.91	[-47.54; -12.28]	1.0%
DePra	8	142.25	31.17	8	147.38	42.81	-5.13	[-41.83; 31.57]	0.2%
DeMscExp3	8	85.29	26.01	8	83.50	33.98	1.79	[-27.86; 31.44]	0.4%
DePhd	5	94.60	13.69	5	108.40	28.50	-13.80	[-41.51; 13.91]	0.4%
Fixed effect model	**135**			**135**			**-1.66**	**[-3.46; 0.14]**	**100%**

Heterogeneity: I-squared=67.7%, tau-squared=19.42, p=0.0017

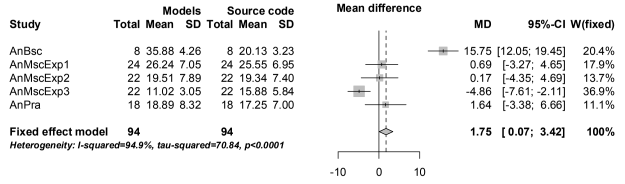
(c) $ab(fb)^{*}c$ - Fixed effect model after excluding studies



	Models			Source code				
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI
DeMscExp1	16	30.06	13.13	16	23.00	7.61	7.06	[-0.38; 14.50]
AnMscExp1	24	26.24	7.05	24	25.55	6.95	0.69	[-3.27; 4.65]
AnMscExp2	22	19.51	7.89	22	19.34	7.40	0.17	[-4.35; 4.69]
AnMscExp3	22	11.02	3.05	22	15.88	5.84	-4.86	[-7.61; -2.11]
AnPra	18	18.89	8.32	18	17.25	7.00	1.64	[-3.38; 6.66]
DeMscExp2	12	157.67	26.11	12	187.58	16.99	-29.91	[-47.54; -12.28]
DePra	8	142.25	31.17	8	147.38	42.81	-5.13	[-41.83; 31.57]
DeMscExp3	8	85.29	26.01	8	83.50	33.98	1.79	
DePhd	5	94.60	13.69	5	108.40	28.50	-13.80	
Random effects model	**135**			**135**			**-1.31**	**[-5.50; 2.88]**

Heterogeneity: I-squared=67.7%, tau-squared=19.42, p=0.0017

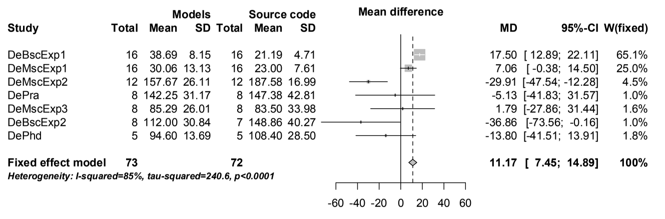
(d) $ab(fb)^{*}d$ - Random effects model after excluding studies



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(fixed)
AnBsc	8	35.88	4.26	8	20.13	3.23	15.75	[12.05; 19.45]	20.4%
AnMscExp1	24	26.24	7.05	24	25.55	6.95	0.69	[-3.27; 4.65]	17.9%
AnMscExp2	22	19.51	7.89	22	19.34	7.40	0.17	[-4.35; 4.69]	13.7%
AnMscExp3	22	11.02	3.05	22	15.88	5.84	-4.86	[-7.61; -2.11]	36.9%
AnPra	18	18.89	8.32	18	17.25	7.00	1.64	[-3.38; 6.66]	11.1%
Fixed effect model	**94**			**94**			**1.75**	**[0.07; 3.42]**	**100%**

Heterogeneity: I-squared=94.9%, tau-squared=70.84, p<0.0001

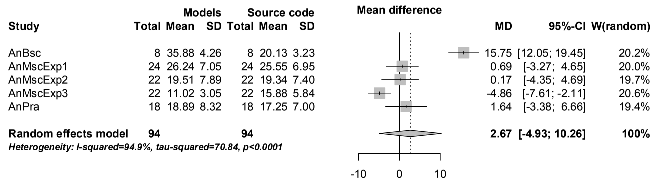
(e) $ab(gb)^{*}c$ - Fixed effect model on the sub-group An



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(fixed)
DeBscExp1	16	38.69	8.15	16	21.19	4.71	17.50	[12.89; 22.11]	65.1%
DeMscExp1	16	30.06	13.13	16	23.00	7.61	7.06	[-0.38; 14.50]	25.0%
DeMscExp2	12	157.67	26.11	12	187.58	16.99	-29.91	[-47.54; -12.28]	4.5%
DePra	8	142.25	31.17	8	147.38	42.81	-5.13	[-41.83; 31.57]	1.0%
DeMscExp3	8	85.29	26.01	8	83.50	33.98	1.79	[-27.86; 31.44]	1.6%
DeBscExp2	8	112.00	30.84	7	148.86	40.27	-36.86	[-73.56; -0.16]	1.0%
DePhd	5	94.60	13.69	5	108.40	28.50	-13.80	[-41.51; 13.91]	1.8%
Fixed effect model	**73**			**72**			**11.17**	**[7.45; 14.89]**	**100%**

Heterogeneity: I-squared=85%, tau-squared=240.6, p<0.0001

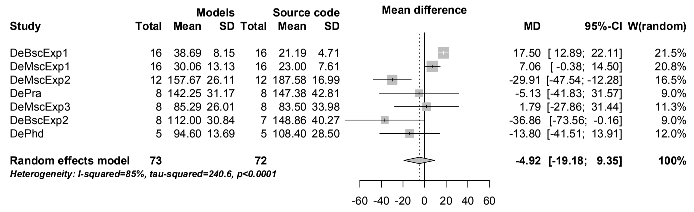
(f) $ab(gb)^{*}c$ - Fixed effect model on the sub-group De



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(random)
AnBsc	8	35.88	4.26	8	20.13	3.23	15.75	[12.05; 19.45]	20.2%
AnMscExp1	24	26.24	7.05	24	25.55	6.95	0.69	[-3.27; 4.65]	20.0%
AnMscExp2	22	19.51	7.89	22	19.34	7.40	0.17	[-4.35; 4.69]	19.7%
AnMscExp3	22	11.02	3.05	22	15.88	5.84	-4.86	[-7.61; -2.11]	20.6%
AnPra	18	18.89	8.32	18	17.25	7.00	1.64	[-3.38; 6.66]	19.4%
Random effects model	**94**			**94**			**2.67**	**[-4.93; 10.26]**	**100%**

Heterogeneity: I-squared=94.9%, tau-squared=70.84, p<0.0001

(g) $ab(gb)^{*}d$ - Random effects model on the sub-group An



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(random)
DeBscExp1	16	38.69	8.15	16	21.19	4.71	17.50	[12.89; 22.11]	21.5%
DeMscExp1	16	30.06	13.13	16	23.00	7.61	7.06	[-0.38; 14.50]	20.8%
DeMscExp2	12	157.67	26.11	12	187.58	16.99	-29.91	[-47.54; -12.28]	16.5%
DePra	8	142.25	31.17	8	147.38	42.81	-5.13	[-41.83; 31.57]	9.0%
DeMscExp3	8	85.29	26.01	8	83.50	33.98	1.79	[-27.86; 31.44]	11.3%
DeBscExp2	8	112.00	30.84	7	148.86	40.27	-36.86	[-73.56; -0.16]	9.0%
DePhd	5	94.60	13.69	5	108.40	28.50	-13.80	[-41.51; 13.91]	12.0%
Random effects model	**73**			**72**			**-4.92**	**[-19.18; 9.35]**	**100%**

Heterogeneity: I-squared=85%, tau-squared=240.6, p<0.0001

(h) $ab(gb)^{*}d$ - Random effects model on the sub-group De

Fig. 3: Forest plots for Completion Time

As the squares in the figure suggest, the experiments contributed equally to the overall result, that is, the use of models slightly improved source-code comprehensibility (MD = 0.01). Indeed, source-code comprehensibility was not significantly different[11] when using or not using models in the execution of comprehension tasks. This result is also confirmed by the overall 95% confidence interval[12] (IC) whose value is $[-0.04; 0.06]$. If we decide to ignore heterogeneity and apply a fixed effects model, we obtain the MD values shown in Figure 2(a). As we can see, the result is quite similar when taking into account the MD values. However, the IC value is $[0.00; 0.05]$.

With regard to Completion time, in Figure 3(a) (sentence abc) it will be noted that, according to the Cochran's Q test, the experiments are heterogenous ($p < 0.0001$). By incorporating heterogeneity and applying a random effects model we obtain an MD value equals to 0.8 and the IC value is $[-5.86; 7.46]$. Unlike Comprehension, the squares are not proportional in size when studying Completion time (see Figure 3(b) — sentence abd). This signifies that some experiments contributed to the overall result more than others. The forest plot suggests that the difference in the completion time is not significant when using or not using models to accomplish comprehension tasks. However, when ignoring heterogeneity and applying a fixed effects model (see Figure 3(a) — sentence abc), the completion time is statistically different when using or not using models in comprehension tasks. The MD value is 3.33 while the IC value is $[1.81; 4.86]$. The choice of how to manage heterogeneity, therefore, proves that the results are crucial in this case, and influences the overall results regarding the impact of UML models on source code comprehensibility.

Since our experiments were heterogeneous with regard to the two dependent variables, we also decided to explore heterogeneity and perform sub-group analyses. In our study, for both the dependent variables introduced in the design of the study (see Section 4.3), we can group experiments according to the kind of models: An and De. We postulated (on the basis of the results of the primary studies) that models produced in the design phase are closer to source code than those produced in the analysis phase. In other words, we could expect that De would aid source code comprehensibility, while An would not. The forest plot for An and comprehension is shown in Figure 2(e). We used a fixed effects model, for the sentence $\underline{abgbc}$, because of the results of the heterogeneously analysis. That is, it can be considered that the studies are not heterogeneous since the Cochran's Q test returned 0.0156 as the value for p (this is why the the sentence of the regular expression aforementioned is underlined). The MD value obtained is low (-0.06) and the IC value is $[-0.11; -0.01]$. It would appear that the presence of UML analysis models does not aid source-code comprehensibility. The assumption made in order to explore heterogeneity and stated above was, therefore in some respects confirmed. The observed results thus allow us to state that UML analysis models focus on the problem domain of the software (the environment in which the software will work) and do not provide any support as regards performing comprehension tasks on source code. Indeed, this kind of models could have confused the participants while comprehending the source code. For example, it could be possible that the participants trusted the models and did not pay adequate attention to the source code. These results are, perhaps, not overly surprising, but they are acceptable, as evidence/postulations need to be empirically verified and/or reaffirmed through the use of empirical studies [6,43,59].

Figure 2(f) shows the forest plot of $\underline{abgbc}$ for the variable Comprehension in the experiments in which the participants were provided with design models (De). The experiments

[11] Effect size is statistically different from the overall effect if the diamond (at the bottom of the forest plot) does not intersect the vertical line.

[12] This is a range of values that we are 95% certain that it contains the true mean value.

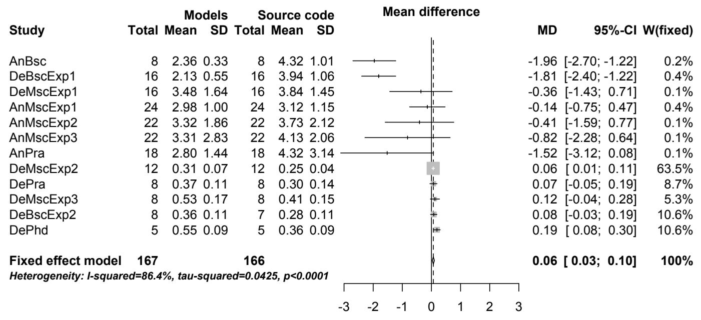
are not heterogeneous since the Cochran's Q test returned a value greater than 0.1 (i.e., $p = 0.1666$). We applied a fixed effects model, since the experiments can be considered homogeneous and it is for this reason that the sentence of the regular expression is underlined. Some experiments contributed to the overall result more than others. The most remarkable outcome is that the difference between using or not using models is significant. Those participants provided with design models understood the source code better because the diamond is on the right-hand side of the vertical line. The MD value is sufficiently large (i.e., 0.06). It is worth mentioning that only for the sentence $abgbc$ for the variable Comprehension and the groups An and De, we observed an homogeneity of the experiments and applied a fixed effects model (i.e., $\underline{abgbc}$).

With regard to Completion time, the meta-analysis results for the An and De subgroups (sentences $abgbc$ and $abgbd$) are summarized in Figure 3(e) and Figure 3(f) and Figure 3(g) and Figure 3(h), respectively. Note that the experiments were heterogenous (see the Cochran's Q test, which returned values less than 0.1 in all the cases). It was for this reason that we applied both the fixed- and random- effects models, i.e., we ignored and incorporated heterogeneity, respectively. The plots shown in Figure 3(g) and Figure 3(h) suggest that the time taken to complete a task was not significantly different when using or not using An and De models and exploiting a random-effects model. More specifically, and on the basis of the descriptive statistics, we can deduce that the participants provided with analysis models needed slightly more time to accomplish a comprehension task. Those in the other experiments spent less time when accomplishing the task with design models. In other words, it would appear that the use of design models paid back the time needed to read them because the effort required to comprehend source code decreased when compared with the effort of the participants provided with only source code. It is also worth mentioning that the exploration of the heterogeneity indicated that the groups of experiments in An and De are not heterogeneous for Comprehension, while they are heterogeneous for Completion time. This confirms that we have presented in Section 3.3, i.e., heterogeneity is not only linked to the kind and type of primary studies but also to the effect measure.

When exploring heterogeneity, one alternative to sub-group analysis is that of carrying out experiment cleaning. We considered the participants' experience in order to exclude experiments. We excluded AnBsc, DeBscExp1, and DeBscExp2 because the participants in these experiments were Bachelor's degree students. The forest plots for Comprehension shown in Figure 2(c) and Figure 2(d) show that the Cochran's Q test suggests that the experiments were heterogeneous since the value of p is less than 0.0001 (the Tau-squared and I-squared values indicated a good extent of such heterogeneity). That is, the experiments were still heterogeneous after this cleaning. It would appear that the participants' experience was not a cause of heterogeneity. We, therefore, incorporated heterogeneity and applied a random effects model (i.e., that shown in Figure 2(d)). It is easy to observe that the model is quite similar to that shown in Figure 2(b), thus confirming the analysis performed when deciding to incorporate heterogeneity. Indeed, source-code comprehensibility was not significantly different when using or not using models, and the MD value obtained is 0.01 while the IC value is $[-0.06; 0.08]$.

Similar to that which occurred with Comprehension, the forest plots for Completion time (see Figure 3(c) and Figure 3(d)) suggest that we can consider the experiments to be heterogeneous. Indeed, the Cochran's Q test suggests that the experiments were heterogeneous (p <0.1) and the Tau-squared and I-squared values indicated a good extent of this heterogeneity. We, then, incorporated heterogeneity and applied a random effects model. Figure 3(d) shows that the squares are not proportional in size, i.e., some experiments contributed to the overall result more than others. Moreover, the forest plot suggests that the difference in the completion time is not significant when using or not using models, and the MD value obtained is -1.31 while the IC value is $[-5.50; 2.88]$.

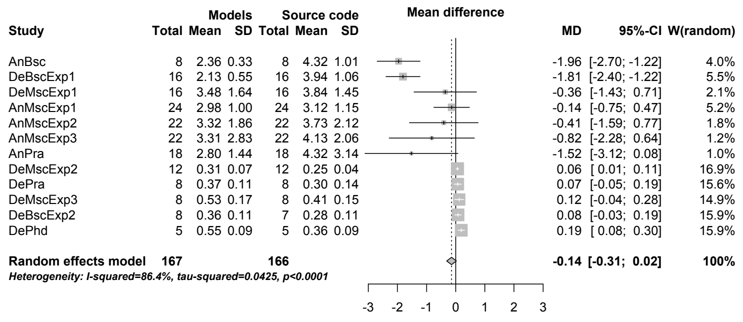
The process shown in Figure 1 was also use to analyze the case of: (i) not exploring heterogeneity and (ii) changing the effect measure by exploiting Efficiency (see Table 3). The results of the Cochran's Q test (p=0.0001) shown in Figure 4(a) and Figure 4(b) suggest that the experiments were heterogeneous. The results of the I-squared indicated a considerable heterogeneity (86.4%). This contrasts with the results of the Tau-squared measure, which suggests that the groups of experiments are not heterogeneous. We considered that experiments were heterogeneous owing to the results of the Cochran's Q test. We, therefore, incorporated heterogeneity and applied a random-effects model. The results obtained are summarized in Figure 4(b). Note that the squares are not proportional in size, i.e., some experiments contributed to the overall result more than others. Moreover, the forest plot suggests that efficiency is not statistically different when using or not using models, and the MD value obtained is -0.14 while the IC value is $[-0.31; 0.02]$. Conversely, upon ignoring heterogeneity and applying a fixed effects model there is a statistically significant difference when using or not using models (see Figure 4(a)). The MD value obtained is 0.06 and the IC value is $[0.03; 0.10]$. As in the case of abc and abd in Figure 3, the choice of how to manage heterogeneity proves to be crucial and can influence the analysis of the impact of models.



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(fixed)
AnBsc	8	2.36	0.33	8	4.32	1.01	-1.96	[-2.70; -1.22]	0.2%
DeBscExp1	16	2.13	0.55	16	3.94	1.06	-1.81	[-2.40; -1.22]	0.4%
DeMscExp1	16	3.48	1.64	16	3.84	1.45	-0.36	[-1.43; 0.71]	0.1%
AnMscExp1	24	2.98	1.00	24	3.12	1.15	-0.14	[-0.75; 0.47]	0.4%
AnMscExp2	22	3.32	1.86	22	3.73	2.12	-0.41	[-1.59; 0.77]	0.1%
AnMscExp3	22	3.31	2.83	22	4.13	2.06	-0.82	[-2.28; 0.64]	0.1%
AnPra	18	2.80	1.44	18	4.32	3.14	-1.52	[-3.12; 0.08]	0.1%
DeMscExp2	12	0.31	0.07	12	0.25	0.04	0.06	[0.01; 0.11]	63.5%
DePra	8	0.37	0.11	8	0.30	0.14	0.07	[-0.05; 0.19]	8.7%
DeMscExp3	8	0.53	0.17	8	0.41	0.15	0.12	[-0.04; 0.28]	5.3%
DeBscExp2	8	0.36	0.11	7	0.28	0.11	0.08	[-0.03; 0.19]	10.6%
DePhd	5	0.55	0.09	5	0.36	0.09	0.19	[0.08; 0.30]	10.6%
Fixed effect model	**167**			**166**			**0.06**	**[0.03; 0.10]**	**100%**

Heterogeneity: I-squared=86.4%, tau-squared=0.0425, p<0.0001

(a) $abeabc$ - Fixed effect model obtained when changing the effect measure and considering all the studies



	Models			Source code					
Study	Total	Mean	SD	Total	Mean	SD	MD	95%-CI	W(random)
AnBsc	8	2.36	0.33	8	4.32	1.01	-1.96	[-2.70; -1.22]	4.0%
DeBscExp1	16	2.13	0.55	16	3.94	1.06	-1.81	[-2.40; -1.22]	5.5%
DeMscExp1	16	3.48	1.64	16	3.84	1.45	-0.36	[-1.43; 0.71]	2.1%
AnMscExp1	24	2.98	1.00	24	3.12	1.15	-0.14	[-0.75; 0.47]	5.2%
AnMscExp2	22	3.32	1.86	22	3.73	2.12	-0.41	[-1.59; 0.77]	1.8%
AnMscExp3	22	3.31	2.83	22	4.13	2.06	-0.82	[-2.28; 0.64]	1.2%
AnPra	18	2.80	1.44	18	4.32	3.14	-1.52	[-3.12; 0.08]	1.0%
DeMscExp2	12	0.31	0.07	12	0.25	0.04	0.06	[0.01; 0.11]	16.9%
DePra	8	0.37	0.11	8	0.30	0.14	0.07	[-0.05; 0.19]	15.6%
DeMscExp3	8	0.53	0.17	8	0.41	0.15	0.12	[-0.04; 0.28]	14.9%
DeBscExp2	8	0.36	0.11	7	0.28	0.11	0.08	[-0.03; 0.19]	15.9%
DePhd	5	0.55	0.09	5	0.36	0.09	0.19	[0.08; 0.30]	15.9%
Random effects model	**167**			**166**			**-0.14**	**[-0.31; 0.02]**	**100%**

Heterogeneity: I-squared=86.4%, tau-squared=0.0425, p<0.0001

(b) $abeabd$ - Random effects model obtained when changing the effect measure and considering all the studies

Fig. 4: Forest plots for Efficiency

Table 4: Summary of results. Values for MD and CI are reported only in the case of a significant effect of the method

Sentence	Dependent Variable	Effect of the method	MD	95%-CI
abc	Comprehension	No	-	-
abd	Comprehension	No	-	-
$\underline{abgbc}$	Comprehension	Yes for both An and De	-0.06; 0.06	[-0.11; -0.01]; [0.03; 0.09]
$\overline{abfbd}$	Comprehension	No	-	-
abc	Completion time	Yes	3.33	[1.81; 4.86]
abd	Completion time	No	-	-
$abgbc$	Completion time	No	-	-
$abgbd$	Completion time	No	-	-
$abfbd$	Completion time	No	-	-
$abeabc$	Efficiency	Yes	0.06	[0.03; 0.1]
$abeabd$	Efficiency	No	-	-

5.2 Discussion

Table 4 summarizes the results of our meta-analysis. For each dependent variable, we show the effect of the method according to each case considered in our analysis (i.e., some of the paths of the process shown in Figure 1, illustrating the sentences of the regular expression defined in Section 3.4) and the corresponding MD and CI values obtained from the meta-analysis. The results suggest that the participants obtained slightly better scores for comprehension when using analysis and design models (the effect of the method is not significant). We can, therefore, conclude that models do not help a lot participants to comprehend source code, although these models do provide additional information on the subject application. In addition, the participants spent more time comprehending source code. This could be related to the effort needed to infer the additional information provided by the models that was definitively not useful as regards attaining an improved comprehension of source code. We further investigated this aspect by performing sub-group analyses, excluding cases, and changing the dependent variable. That is, we looked at the possible reasons why the experiments were heterogeneous. The most plausible justification was: analysis models refer to objects (or entities) in the problem domain, while design models refer to objects in the solution domain and should, therefore, better support source-code comprehension tasks. The results obtained from meta-analyses of the two sub-groups chosen gave credit to our assumption. In particular, the results indicate that the use of models affects source-code comprehensibility, but in two slight different directions. The use of analysis models reduces source-code comprehensibility (see CI in Table 4) and increases the time taken to complete comprehension tasks, while the use of design models improves source-code comprehensibility (see CI in Table 4) and reduces task completion time. The MD values for source-code comprehensibility for the analysis and design models are -0.06 and 0.06, respectively. This outcome should not be at all surprising because analysis and design models are created for different purposes, although companies seem to ignore this difference [56] as also we discussed in the introductive part of Section 4. Analysis models are created to capture a domain, to represent a set of requirements, to understand a poorly focused problem boundary, and so on, while design models can be used to structure source code and capture design artifacts that do not directly emerge from requirements [10]. As such, analysis models say little or nothing about source code and the use of this kind of models will not, therefore, benefit comprehension. For example, in Figure 5 we show some models of the Music Shop experimental object (i.e., that used in the experiments carried out by An group). It is easy to see that these models do not provide implementation details, even if some of these details and some design decisions could be inferred. For example, some of the classes to be understood in Figure 5(b) are in the source code (not reported here owing to their scant relevance, but available on the web for download) and also in the class diagram (available on the web for the download) in the experiments of the De group. That is, some classes in the problem domain are present in both the solution domain and the source code, thus possibly allowing the participant to also obtain a little information on the implementation from the analysis models. This scenario is customary for more traditional development approaches (e.g., Unified Software Development Process) [10]. With regard to an example of design decision, the architectural pattern used could be used in an inferred manner. In particular, we could postulate that the architectural pattern implemented in Music Shop is the Model-View-Controller, given the division of the classes in: boundary, control, and entity. In both the cases mentioned above, it is the developers' ability (and possibly his/her knowledge of the subject software and its domain) that could make the difference in terms of source-code comprehension, rather than the actual information the analysis models provide. In summary, we can assume that design models help more than analysis models in the comprehension of source code, since they are focused on the implementation aspects. On the basis of our considerations, we can summarize our research results as follows:

- *Software models produced in the design phase aid in source-code comprehensibility.*

Despite the fact that our results improved the findings obtained for the individual experiments conducted in our long-term research, we cannot provide conclusive findings concerning whether analysis models helped in the understanding of source code in the context of graduate, undergraduate, PhD students, and novice practitioners.

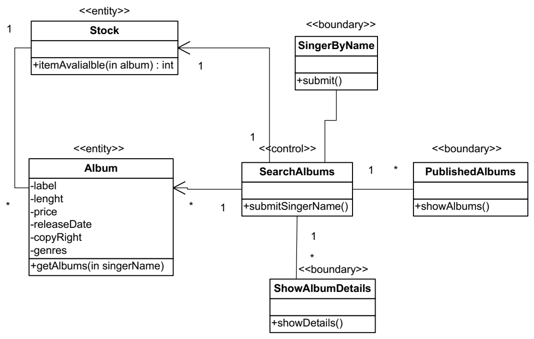
5.3 Implications

We judged the implications of our study by adopting a perspective-based approach [7] and discuss these implications on the basis of the perspectives of practitioner/consultant (from here on simply practitioner) and researcher [41]. When applicable, we also discuss future research directions related to these implications.

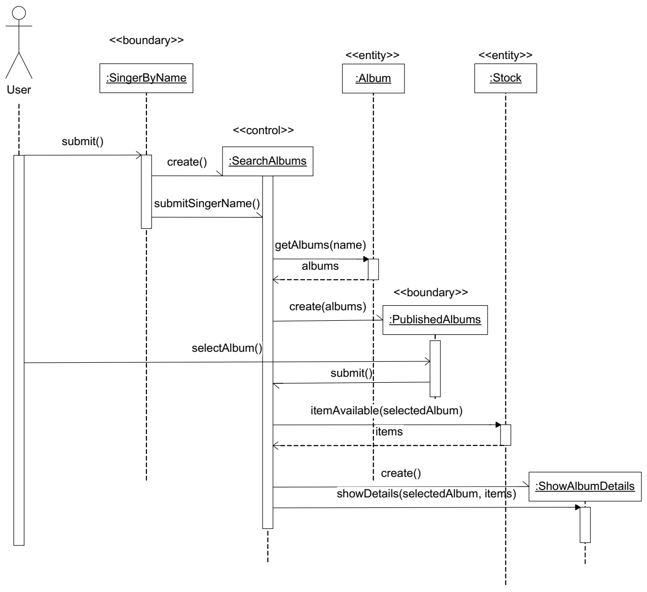
- UML-based modeling is important as it allows an improved comprehension of source code. These models should focus on aspects related to the solution domain (i.e., implementation aspects) of a subject software. Models produced in the analysis phase could, therefore, be considered of less importance if they are only intended to support the comprehension of source code. Furthermore, these models are of primary importance when they contain the subsequent development phases. We can speculate on this point because we used the same software in some of the experiments, but the models were produced in either the analysis or the design phases (e.g., AnBsc and DeMscExp1). From the practitioner's perspective, this result is relevant because it could be useful to adopt a development process based on the use of the UML. It might, however, be useless to give UML-based analysis models to the software engineer when he/she has to perform small maintenance operations on source code. This is aligned with the findings of [5, 15] (see Section 2) whose authors stated that the UML only seemed to be really useful as regards understanding complex systems. That is, this kind of models should be only used to support the subsequent phases of the development or to improve the comprehension of

Use Case Name: *Search Album by Singer*
ID: 3
Brief description of the use case: The user selects an album by specifying the singer's name. Details of the selected album will be shown by the system (e.g., original release date)
Main Actors: User
Flow of events: 1. The User inserts the singer name. 2. The system presents the album list of the specified singer. 3. The user selects the album of interest. 4. The system shows details on the chosen album and the number of available copies in the stock. As result, the presented information is: original release date, label, copyright, genres, length, price, and number of available copies.
Pre-condition: The user has selected the functionality search singer by name.
Post-condition: 1. The system shows details about a given album for the chosen singer.

(a) The *Search Album by Singer* Use Case



(b) Class diagram



(c) Sequence diagram for the functionality *Search Album by Singer*

Fig. 5: Some of the analysis models for the Music Shop application

functional requirements [1]. From the researcher's perspective, it would be interesting

to investigate whether variations in the context (e.g., larger systems and more or less experienced software engineers) might lead to different results.
- UML analysis models appear to uselessly overload participants when performing comprehension tasks. Once again, the results obtained in our study coincide with those from some of the related work (e.g., [20,22]) in which only design models appeared to help achieving a better understanding of the systems. This result is relevant for the researcher because it would be interesting to carry out further research into this aspect and discover in which context it holds.
- Although we are not sure whether our findings scale up to real projects, the results obtained could be true in all those cases in which the models are concerned with a part of the entire software and maintenance operations are performed on a chunk of the source code of the entire system.
- We observed that the models produced in the design phase aid the comprehension of source code and postulated that this is because they are closer to source code than those produced in the analysis phase. It might, therefore, be expected that reverse-engineered diagrams could also be at least as effective as the forward designed diagrams as regards aiding source code comprehension since they are obtained directly from the source code. Our results and those of Fernández-Sáez et. al. presented in [22] provide the basis for future work in this direction. This point is clearly relevant for any researchers who might be interested in studying the delineated research direction. If future work confirms that reverse-engineered diagrams are as effective as forward ones, the practitioner could be further motivated to use reverse engineering tools in his/her company.
- We focused on desktop applications. The models of these systems were sufficiently realistic for small-sized in-house software and subcontracting development projects. From the researcher's perspective, the effect of analysis and design models on different types of systems (e.g., smartphone and web apps) represents a possible future direction for our research. This point is clearly relevant for the researcher.
- The UML is widely used in the software industry. The results achieved are, therefore, useful for all the companies that exploit the UML as a support when software engineers are executing comprehension tasks (e.g., performing maintenance/evolution operations). Studies on this notation are currently required to understand the cases in which its use improves the comprehensibility and maintainability of source code. There are currently only a few evaluations, as we have discussed in the related work section.
- Dealing with heterogeneity could be crucial, since it may influence the results of the meta-analysis. This point is clearly of interest for any researchers who might be interested in studying how to deal with heterogeneity when integrating results from several studies by using a meta-analysis.

5.4 Threats to Validity

Despite our efforts to mitigate as many threats to validity as possible, some are unavoidable. In order to comprehend the strengths and limitations of our empirical study, the threats that could have affected the results are presented and discussed as follows.

5.4.1 Internal Validity.

This kind of validity is of concern when causal relations are examined. In our study, the design of the experiments could have affected the results. Different kinds of design were considered in each experiment. Each group of participants either worked on two different tasks with and without models (maturation or diffusion or imitation of treatments) or worked on a single task either with models or without models. The artifacts used to carry out the experiments (e.g., comprehension questionnaire and documentation) could also have negatively affected the experiments and thus the outcomes of the meta-analysis. We mitigated these threats by accurately designing all the material used in each experiment. In many cases, pilot studies were conducted to assess this experimental material. Threats to Internal validity could also depend on how the participants are selected from a larger group. How the experiments were selected could also have affected the results. With regard to multiple group experiments the results could have been biased because of the different behavior of participants in different groups (i.e., interactions with selection). Social threats to internal validity could also have been present in the experiments.

5.4.2 External Validity.

Performing experiments with students could lead to doubts concerning their representativeness when compared to software professionals (interaction of selection and treatment). In addition to experiments with students, we also carried out experiments with professionals and PhD students. It is worth mentioning that the tasks used did not require a high level of industrial experience. This led us to believe that the use of students was not a major issue here [13]. However, if tasks are too simple they may be not representative. This could imply threats to the validity of the results. Another threat to external validity concerns the experimental objects (interaction of setting and treatment). For example, we removed comments from source code and did not provide any explicit information on the traceability links between models and source code. We took these decisions to avoid the effect of source code comments and traceability links being confused with the main factor studied. With regard to source code comments, some further considerations are required: (i) comments and source code may not be coherent (i.e., the comment does not describe the intent of the method and its actual implementation) with one another [14] and (ii) it is possible that experienced developers (e.g., professionals) do not use comments (because they are often not updated when source code changes [23,39]) while performing comprehension tasks [52]. The first point could be dealt with by modifying the comments to make them coherent with the source code. However, this choice could affect external validity. As for the second point, we can do little or nothing. In fact, it could be that professional developers (and possibly PhD students) do not take much care with the comments, while those with little experience do. We, however, advise the use source code comments and traceability links in future studies. Our research provides the basis for future work on this matter. We also indented the source code when preparing the experimental objects. This design choice could affect external validity. However, many of the available IDEs provide a feature to remove this kind of smell from source code. Therefore, it could happen that some of the participants indent the code, while other no. If this happened a series of threats to the conclusion validity could be risen. That is, if any effect of this kind of smell could be present it could affect results in an undesirable and uncontrollable way.

5.4.3 Construct Validity.

This kind of validity may have been influenced by the measures used to obtain a quantitative evaluation of source-code comprehensibility (inadequate preoperational explanation of constructs). Construct validity might also have been affected by the comprehension used and the post-experiment survey questionnaires, in addition to social threats. We employed post-experiment survey questionnaires designed using standard approaches and scales [47]. The responses to this kind of questionnaire were used to explain the quantitative results. Another threat to external validity is mono-operation bias. All the experiments in our study included a single independent variable (or treatment). This may have under-represented the construct and thus not provided the full picture of the theory. The threat concerning the interaction of different treatments is not present in our research because the participants were involved in only one experiment. In order to mitigate construct validity, we conducted external replications and their results were subsequently aggregated with the other experiments and replications.

5.4.4 Conclusion Validity.

This kind of validity concerns the ability to draw correct conclusions. In order to deal with conclusion validity, we performed a statistical analysis of the data gathered. Despite our effort, threats to low statistical power could still have been present. The number of experiments considered in our study might also have affected the results. With regard to the selection of the populations, we drew fair samples and conducted our experiments with participants belonging to these samples. Another threat to conclusion validity could be related to the number of participants. This kind of threat was mitigated because our study was based on 333 observations (the largest on the UML). The reliability of measures might affect results. In each individual experiment, the experimenters attempted to mitigate this kind of threat as much as possible. The threat related to the random heterogeneity of subjects could have been present in our meta-analysis study and in each experiment. We took this aspect into account in the meta-analysis. Finally, we did not deal with clinical heterogeneity or methodological heterogeneity. As mentioned in Section 3.2, statistical heterogeneity can be considered as a consequence of either, or both, clinical heterogeneity or methodological heterogeneity.

6 Conclusion

In this paper, we have presented the results of long-term research into the effect of using UML-based modeling in software maintenance, and in source-code comprehensibility in particular. This research began in 2009 with an industrial survey [56], and the results of this survey were then used as a basis to conduct a number of controlled experiments (internal and external replications) with students and practitioners from Italy and Spain. The results of individual experiments were synthesized by means of a meta-analyses and presented in this paper. The most important outcome is: the use of UML models is important as regards allowing software engineers to better understand source code, given that these models focus on aspects related to objects (or entities) in the solution domain of a subject software. Models produced in the analysis phase are of less importance if their purpose is solely to enable the comprehension of source code.

Possible future directions for our research are: (i) performing further experimentation considering different and larger software systems related to unknown domains in order to verify whether the findings obtained are still valid; (ii) studying the effect of providing the participants with information in an incremental manner; (iii) analyzing the effect of different UML diagrams; and (iv) investigating the effect of the same UML diagrams on non-source code comprehension tasks.

Acknowledgements The authors would like to thank the participants in the experiments and all the people who supported the research presented in this paper. This work has been partially supported by the SEQUOIA Project, (TIN2015-63502-C3-1-R) (MINECO/FEDER) funded by Fondo Europeo de Desarrollo Regional and Ministerio de Economa y Competitividad

References

1. Abrahão, S.M., Gravino, C., Pelozo, E.I., Scanniello, G., Tortora, G.: Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: Results from a family of five experiments. IEEE Transactions on Software Engineering **39**(3) (2013)
2. Agarwal, R., Sinha, A.P.: Object-oriented modeling with UML: a study of developers' perceptions. Commun. ACM **46**(9), 248–256 (2003)
3. Anda, B., Hansen, K., Gullesen, I., Thorsen, H.K.: Experiences from introducing UML-based development in a large safety-critical project. Empirical Software Engineering **11**(4), 555–581 (2006)
4. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering **28**(10), 970–983 (2002)
5. Arisholm, E., Briand, L.C., Hove, S.E., Labiche, Y.: The impact of UML documentation on software maintenance: An experimental evaluation. IEEE Transactions on Software Engineering **32**(6), 365–381 (2006)
6. Basili, V., Shull, F., Lanubile, F.: Building knowledge through families of experiments. IEEE Transactions on Software Engineering **25**(4), 456–473 (1999)
7. Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, L.S., Zelkowitz, M.V.: The empirical investigation of perspective-based reading. Empirical Software Engineering **1**(2), 133–164 (1996)
8. Basili, V.R., Rombach, H.D.: The TAME project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering **14**(6), 758–773 (1988)
9. Bavota, G., Canfora, G., Di Penta, M., Oliveto, R., Panichella, S.: An empirical investigation on documentation usage patterns in maintenance tasks. In: Proceedings of International Conference on Software Maintenance, pp. 210–219. IEEE Computer Society (2013)
10. Bruegge, B., Dutoit, A.H.: Object-Oriented Software Engineering: Using UML, Patterns and Java, 2nd edition. Prentice-Hall (2003)
11. Budgen, D., Burn, A.J., Brereton, O.P., Kitchenham, B.A., Pretorius, R.: Empirical evidence about the UML: a systematic literature review. Software: Practice and Experience **41**(4), 363–392 (2011)
12. Cariou, E., Beugnard, A., Jezequel, J.M.: An architecture and a process for implementing distributed collaborations. In: Proceedings of International Enterprise Distributed Object Computing, pp. 132–143 (2002)
13. Carver, J., Jaccheri, L., Morasca, S., Shull, F.: Issues in using students in empirical studies in software engineering education. In: Proceedings of International Symposium on Software Metrics, pp. 239–250. IEEE Computer Society (2003)
14. Corazza, A., Maggio, V., Scanniello, G.: Coherence of comments and method implementations: a dataset and an empirical investigation. Software Quality Journal pp. 1–27 (2016)
15. Dzidek, W.J., Arisholm, E., Briand, L.C.: A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Transactions on Software Engineering **34**(3), 407–432 (2008)
16. Eclipse Modeling Framework (EMF): http://www.eclipse.org/modeling/emf/
17. Erickson, J., Siau, K.: Theoretical and practical complexity of modeling methods. Commun. ACM **50**(8), 46–51 (2007)
18. Fernández-Sáez, A.M., Caivano, D., Genero, M., Chaudron, M.R.V.: On the use of UML documentation in software maintenance: Results from a survey in industry. In: Proceedings of ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 292–301 (2015)
19. Fernández-Sáez, A.M., Chaudron, M.R.V., Genero, M.: Exploring costs and benefits of using UML on maintenance: Preliminary findings of a case study in a large IT department. In: Proceedings of the International Workshop on Experiences and Empirical Studies in Software Modeling co-located with the International Conference on Model Driven Engineering Languages and Systems, pp. 33–42 (2013)
20. Fernández-Sáez, A.M., Genero, M., Caivano, D., Chaudron, M.R.V.: Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments. Empirical Software Engineering **21**(1), 212–259 (2016)
21. Fernández-Sáez, A.M., Genero, M., Chaudron, M.R.V.: Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. Information & Software Technology **55**(7), 1119–1142 (2013)

22. Fernández-Sáez, A.M., Genero, M., Chaudron, M.R.V., Caivano, D., Ramos, I.: Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: A family of experiments. Information & Software Technology **57**, 644–663 (2015)
23. Fluri, B., Wursch, M., Gall, H.: Do code and comments co-evolve? on the relation between source code and comment changes. In: Proceedings of the Working Conference on Reverse Engineering, pp. 70–79. IEEE Computer Society (2007)
24. Fu, R., Gartlehner, G., Grant, M., Shamliyan, T., Sedrakyan, A., Wilt, T.J., Griffith, L., Oremus, M., Raina, P., Ismaila, A., Santaguida, P., Lau, J., Trikalinos, T.A.: Conducting quantitative synthesis when comparing medical interventions: AHRQ and the effective health care program. Journal of Clinical Epidemiology **64**(11), 1187 – 1197 (2011)
25. Gamma, E., Helm, R., R.Johnson, Vlissides, J.: Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley (1995)
26. Garousi, G., Garousi, V., Moussavi, M., Ruhe, G., Smith, B.: Evaluating usage and quality of technical software documentation: an empirical study. In: Proceedings of International Conference on Evaluation and Assessment in Software Engineering, pp. 24–35 (2013)
27. Gravino, C., Scanniello, G., Tortora, G.: Source-code comprehension tasks supported by UML design models: Results from a controlled experiment and a differentiated replication. J. Vis. Lang. Comput. **28**, 23–38 (2015)
28. Gravino, C., Tortora, G., Scanniello, G.: An empirical investigation on the relation between analysis models and source code comprehension. In: Proceedings of the International Symposium on Applied Computing, pp. 2365–2366. ACM (2010)
29. Grossman, M., Aronson, J.E., McCarthy, R.V.: Does UML make the grade? Insights from the software development community. Information & Software Technology **47**(6), 383–397 (2005)
30. Guéhéneuc, Y.G.: P-mart: Pattern-like micro architecture repository. In: Proceedings of EuroPLoP Focus Group on Pattern Repositories (2007)
31. Hammad, M., Collard, M.L., Maletic, J.I.: Automatically identifying changes that impact code-to-design traceability during evolution. Software Quality Journal **19**(1), 35–64 (2011)
32. Higgins, J.P.T., Green, S.: Cochrane Handbook for Systematic Reviews of Interventions, 5 edn. The Cochrane Collaboration (2008)
33. Huedo-Medina, T.B., Sánchez-Meca, J., Marín-Martínez, F., Botella, J.: Assessing heterogeneity in meta-analysis: Q statistic or i2 index? Psychol Methods **11**(2), 193–206 (2006)
34. Hutchinson, J.E., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: Proceedings of the International Conference on Software Engineering, pp. 471–480 (2011)
35. ISO: Information Technology–Software Product Evaluation: Quality Characteristics and Guidelines for their Use, ISO/IEC IS 9126. ISO, Geneva (1991)
36. ISO: ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) – Part 9: Requirements for non-keyboard input devices. ISO, Geneva, Switzerland (2000)
37. ISO: ISO/IEC 25010 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. ISO, Geneva, Switzerland (2011)
38. Jedlitschka, A., Ciolkowski, M., Pfahl, D.: Reporting experiments in software engineering. In: Guide to Advanced Empirical Software Engineering (Eds) F. Shull and J. Singer and D. Sjoberg, pp. 201–228. Springer (2008)
39. Jiang, Z.M., Hassan, A.E.: Examining the evolution of code comments in postgresql. In: Proceedings of Mining Software Repositories, pp. 179–180. ACM (2006)
40. Juristo, N., Moreno, A.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers (2001)
41. Kitchenham, B., Al-Khilidar, H., Babar, M., Berry, M., Cox, K., Keung, J., Kurniawati, F., Staples, M., Zhang, H., Zhu, L.: Evaluating guidelines for reporting empirical software engineering studies. Empirical Software Engineering **13**, 97–121 (2008)
42. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering (2007)
43. Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Transactions on Software Engineering **28**(8), 721–734 (2002)
44. Lehnert, S., Farooq, Q.u.a., Riebisch, M.: Rule-based impact analysis for heterogeneous software artifacts. In: Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 209–218 (2013)
45. Leotta, M., Ricca, F., Antoniol, G., Garousi, V., Zhi, J., Ruhe, G.: A pilot experiment to quantify the effect of documentation accuracy on maintenance tasks. In: Proceedings of International Conference on Software Maintenance, pp. 428–431 (2013)

46. OMG: Unified modeling language (UML) specification, version 2.0. Tech. rep., Object Management Group (2005)
47. Oppenheim, A.N.: Questionnaire Design, Interviewing and Attitude Measurement. Pinter, London (1992)
48. Pavalkis, S., Nemuraite, L.: Process for Applying Derived Property Based Traceability Framework in Software and Systems Development Life Cycle, pp. 122–133. Springer Berlin Heidelberg (2013)
49. Pavalkis, S., Nemuraite, L., Butkiene, R.: Derived properties: A user friendly approach to improving model traceability. Information Technology and Control **42**(1), 48–60 (2013)
50. Pickard, L., Kitchenham, B.A., Jones, P.: Combining empirical results in software engineering. Information & Software Technology **40**(14), 811–821 (1998)
51. Ried, K.: Interpreting and understanding meta-analysis graphs - A practical guide, vol. 35. Australian College of General Practitioners (2008)
52. Salviulo, F., Scanniello, G.: Dealing with identifiers and comments in source code comprehension and maintenance: Results from an ethnographically-informed study with students and professionals. In: Proceedings of International Conference on Evaluation and Assessment in Software Engineering. ACM (2014)
53. Scanniello, G., Gravino, C., Genero, M., Cruz-Lemus, J.A., Tortora, G.: On the impact of UML analysis models on source code comprehensibility and modifiability. ACM Transactions on Software Engineering and Methodology **23**(2) (2014)
54. Scanniello, G., Gravino, C., Risi, M., Tortora, G.: A controlled experiment for assessing the contribution of design pattern documentation on software maintenance. In: Proceedings of the Symposium on Empirical Software Engineering and Measurement. ACM (2010)
55. Scanniello, G., Gravino, C., Risi, M., Tortora, G., Dodero, G.: Documenting design-pattern instances: A family of experiments on source-code comprehensibility. ACM Transactions on Software Engineering and Methodology **24**(3), 14 (2015)
56. Scanniello, G., Gravino, C., Tortora, G.: Investigating the role of UML in the software modeling and maintenance - a preliminary industrial survey. In: Proceedings of International Conference on Enterprise Information Systems, pp. 141–148 (2010)
57. Scanniello, G., Gravino, C., Tortora, G., Genero, M., Risi, M., Cruz-Lemus, J.A., Dodero, G.: Studying the effect of uml-based models on source-code comprehensibility: Results from a long-term investigation. In: Springer (ed.) Proceedings of International Conference on Product-Focused Software Process Improvement, vol. 9459, pp. 311–327. Lecture Notes in Computer Science (2015)
58. Settimi, R., Cleland-Huang, J., Khadra, O.B., Mody, J., Lukasik, W., DePalma, C.: Supporting software evolution through dynamically retrieving traces to uml artifacts. In: Proceedings of International Workshop on Principles of Software Evolution, pp. 49–54 (2004)
59. Shull, F., Carver, J.C., Vegas, S., Juzgado, N.J.: The role of replications in empirical software engineering. Empirical Software Engineering **13**(2), 211–218 (2008)
60. Sillito, J., Murphy, G.C., De Volder, K.: Asking and answering questions during a programming change task. IEEE Transactions on Software Engineering **34**(4), 434–451 (2008)
61. Tang, A., Jin, Y., Han, J.: A rationale-based architecture model for design traceability and reasoning. Journal of Systems and Software **80**(6), 918–934 (2007)
62. Tang, A., Nicholson, A., Jin, Y., Han, J.: Using bayesian belief networks for change impact analysis in architecture design. J. Syst. Softw. **80**(1), 127–148 (2007)
63. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer (2012)
64. Zhi, J., Garousi-Yusifoglu, V., Sun, B., Garousi, G., Shahnewaz, S.M., Ruhe, G.: Cost, benefits and quality of software development documentation: A systematic mapping. Journal of Systems and Software **99**, 175–198 (2015)