Pós-Graduação em Ciência da Computação

Waldemar Pires Ferreira Neto

Choose the Middle Way: Supporting Coding Experiments According to Their Particular Context Characteristics

RECIFE

2018

Waldemar Pires Ferreira Neto

Choose the Middle Way: Supporting Coding Experiments According to Their Particular Context Characteristics

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientador: Sergio Castelo Branco Soares
Maria Teresa Baldassarre

RECIFE
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

F383c Ferreira Neto, Waldemar Pires
Choose the middle way: supporting coding experiments according to their particular context characteristics / Waldemar Pires Ferreira Neto. – 2017.
190 f.: il., fig., tab.

Orientador: Sérgio Castelo Branco Soares.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências e apêndice.

1. Engenharia de software. 2. Experimentação. I. Soares, Sérgio Castelo Branco (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2018-085

Waldemar Pires Ferreira Neto

Choose the Middle Way: Supporting Coding Experiments According to Their Particular Context Characteristics

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação

Science is organized knowledge. Wisdom is organized life. — Immanuel Kant

AGRADECIMENTOS

Nessa caminhada, agradeço a tantos lugares,
Às cidades que me acolheram, apesar dos pesares
Cidades são as pessoas os principais pilares
São elas que te dão as cores e os ares

Nessa caminhada, agradeço também à família,
As que compartilho ou compartilhei da homilia
Dizem que se escolhem e como se enganam
Elas que lhe acolhem e as únicas que no fim amam

Nessa caminhada agradeço aos tutores,
Curadores sublimes, mais que coautores
Apesar das minhas falhas e imperfeição
Mostraram que meu caminho é com pés no chão

Nessa caminhada, agradeço aos amigos
Sem nenhuma distinção, sejam novos ou antigos
Pessoas inestimáveis, talvez não os mereça
Sempre estão lá no momento que careça

Nessa caminhada, agradeço aos colegas
Amigos ou colegas? Uma escolha as cegas
Alegria num café ou num mero bom dia
Elas não percebem a luz que irradia

Nessa caminhada, agradeço aos amores
Quase não diferencio "tu chegas" e "tu fores"
Ensinaram que o essencial voltar-me pra mim
Apreciar o Damasco ou o cheiro do Jasmim

E para cada página que daqui se segue
Mostrarei o resultado como demando e entregue
Mas talvez seja nessa página escrita a improviso
Talvez mostro o maior resultado, e logo aviso

“Eu acredito, que às vezes são as pessoas que ninguém espera nada que fazem as coisas que ninguém consegue imaginar."

(Alan Turing)

ABSTRACT

Experiments play an essential role in evaluating solutions in software engineering. A field of software engineering where experiments are frequently used is software development. In this field, many solutions are proposed to foster coding activities, such as different programming languages, developing techniques, tools, and other solutions. In this context, this research has two primary goals. The first goal is to investigate experiments performing coding activities (Coding Experiments). This investigation shall raise the most common characteristics of such experiments, and how current solutions supporting experiments address such coding context. The second goal is to propose a solution to support coding experiments according to their particular context characterization. This research was divided in many sub-steps. Each sub-step adopted distinct method. In the first step, we conducted a systematic analysis of coding experiments published in seven renowned venues in software engineering from 2003 to 2016. In the next step, we systematically evaluated the current solutions to support experiments in SE according to previous findings. Based on the results from previous steps, we proposed our solution. Finally, in the final step, we carried out a case study on replicating coding experiments with the proposed solution. The first study revealed many issues in coding experiments that can be addressed to aid its execution. The second study brought to light which aspects of coding experiments are covered by current solutions. In fact, the majority of general characteristics of coding experiments are adequately addressed by current solutions. However, some context-specific characteristics are not satisfactorily undertaken. Based on found lacks, a metamodel was proposed to specify context-specific coding experiments characteristics. This metamodel can be seen as a specialization of current solution focusing only on coding context characterization. Also, a set of tools were developed to (i) specify models according to the proposed metamodel and (ii) support the experiment execution according to its specification. The last study assessed the proposed solution usage to help researchers carrying out coding activities in experiments. From experiment planner's perspective, the effort to conduct and collect data was reduced, even considering the extra effort to specify the coding experiment. From participants' perspective, the proposed solution seemed reasonable to support experiments. However, some issues were identified. Finally, although positive results, performing more assessments including different settings is required to generalize these results. This research focuses only on supporting coding experiments, more precisely planning and execution phases. This work does not deal with other aspects such as data analysis, and we believe current solutions can deal with them. By specifying coding context characteristics, many activities when carrying out a coding experiment can be (semi-)automated, thus contributing to reduce effort to experiment. Moreover, the proposed solution proved adequate for supporting coding experiments, and it is available to support researches around the world through our repository.

Key-words: Software Engineering. Coding Experimentation. Model-driven Environment.

RESUMO

Experimentos desempenham um papel essencial na avaliação de soluções em engenharia de software (ES). Um campo da ES onde experimentos são frequentemente utilizados é o de desenvolvimento de software. Nele, soluções são propostas para facilitar ou melhorar as atividades de codificação, como diferentes linguagens de programação, desenvolvimento de técnicas, ferramentas e outras soluções. Neste contexto, esta pesquisa tem dois objetivos principais. O primeiro é investigar os experimentos que realizam atividades de codificação (Experimentos de Codificação). Esta investigação deve levantar as características mais comuns de tais experimentos, bem como as soluções atuais que auxiliam a sua realização. O segundo objetivo é propor uma solução para auxiliar estes experimentos de acordo com suas características particulares. Foram adotadas várias metodologias para cada fase da pesquisa. No primeiro estudo, adotamos uma análise sistemática de experimentos de codificação publicados em sete renomados fóruns científicos na engenharia de software entre 2003 a 2016. Depois, realizamos uma avaliação sistemática das atuais soluções para apoiar experimentos em SE. Baseado nos resultados da fases anteriores, propusemos nossa solução. E por fim, conduzimos um quasi-experimento com a solução proposta. Nesse experimento, replicamos um experimento externo que envolvia codificação. O primeiro estudo revelou algumas limitações na realização de experimentos de codificação. Tais limitações podem ser exploradas com o intuito de auxiliar a execução destes experimentos. O segundo estudo identificou lacunas nas atuais soluções que dão apoio a realização de experimentos na engenharia de software. Com base nos resultados anteriores, foi proposto um metamodelo para explicitar as características dos experimentos de codificação que sejam específicas para os seus contextos particulares. Este metamodelo pode ser visto como uma especialização das soluções atuais, focado apenas na caracterização do contexto de codificação. Além disso, um conjunto de ferramentas foi desenvolvido para (i) especificar modelos de acordo com o metamodelo proposto e (ii) apoiar a execução dos experimentos de acordo com a especificação. O último estudo avaliou o uso da solução proposta para auxiliar pesquisadores a realizar experimentos de codificação. Do ponto de vista de um pesquisador, o esforço para conduzir e coletar dados foi reduzido, mesmo considerando o esforço extra para modelar o experimento de codificação. Do ponto de vista dos participantes, a solução proposta aparenta ser útil para apoiar a realização do experimento. No entanto, alguns problemas foram identificados. Finalmente, apesar de resultados positivos, ainda se faz necessário à realização de mais avaliações, incluindo configurações diferentes, para podermos generalizar os resultados obtidos. Esta pesquisa concentra-se apenas no apoio a realização de experimentos de codificação, e mais precisamente as fases de planejamento e execução. Este trabalho não trata de outros aspectos, como análise de dados, e acreditamos que soluções atuais são satisfatórias para lidar com esses outros aspectos. Ao especificar características de contexto de experimento de codificação, muitas atividades do experimento podem ser (semi-) automatizadas, contribuindo assim para reduzir o esforço de experimentação. Além disso, a solução proposta mostrou-se adequada para auxiliar experimentos de codificação, e está disponível para apoiar pesquisas pelo mundo através do nosso repositório.

Palavras-chaves: Engenharia de Software. Experimentação. Experimento de Codificação. Engenharia baseada em Modelos.

List of Figures

List of Tables

Contents

1 INTRODUCTION

To apply an experimental test would be to show ignorance of the difference between human nature and divine.

—Plato, *Timaeus and Critias*

Since the dawn of science, philosophers placed great emphasis on direct observation of nature (ARISTOTLE, 350BCE; PLATO, 2012). They taught us about the importance of developing theories based on facts. Many "natural sciences" (such as physics, biology, and medicine) have followed this principle for centuries. In such sciences, empirical studies are a useful way to observe "natural facts" and draw conclusions from them.

Not only in "natural sciences" empirical studies play an important role. Other fields of knowledge, where we are unable to state any laws of nature like social sciences and psychology, have a tradition in empirical studies. Moreover, such sciences are concerned with human behavior. An important observation, in this context, is hence that software engineering is very much governed by human behavior through the people developing software. Thus, we cannot expect to find any formal rules or laws in software engineering except perhaps when focusing on specific technical aspects.

It may be helpful to discuss the why experimentation in software engineering briefly before going further. The primary reason for carrying out an experiment is the opportunity of getting measurable and statistically significant results concerning the understanding, controlling, prediction, and improvement of software development. In this context, experiments study more than one treatment to analyze their outcomes. For instance, if it is possible to control who is applying one technique and who is applying another, and when and where they are used, then it is possible to perform an experiment. Such manipulations can be made in an off-line situation, for example in a laboratory (under controlled circumstances), where the events are organized to simulate their appearance in the real world. On the other hand, experiments may alternatively be made online, which means that the investigation is executed in the field in a real-life context (WOHLIN et al., 2012).

Regarding software engineering, Wohlin et al. (2012) classify experiment in software engineering into two categories: human-oriented and technology-oriented experiments. In human-oriented experiments, human beings apply different treatments to objects, while in technology-oriented experiments, various technical treatments are applied to distinct objects. According to Kitchenham et al. (2013), human-oriented experiments are often challenging due to human factor involved in such experiments.

1.1 PROBLEM OVERVIEW

According to Wohlin et al. (2012), the very first human-oriented experiments in software engineering were conducted in the late 1960s by Sackman, Erikson e Grant (1968) about online and offline work in testing. In the middle of the 1980s, Basili, Selby e Hutchens (1986) exposed the need for systematic experiments in software engineering. Other articles stressing the need for empiricism in software engineering were published later (BASILI, 1993; FENTON; PFLEEGER; GLASS, 1994; GLASS, 1994; KITCHENHAM; PICKARD; PFLEEGER, 1995; POTTS, 1993; TICHY, 1998). The lack of empirical evidence in software engineering research was emphasized by Tichy et al. (1995), Zelkowitz e Wallace (1998), Glass, Vessey e Ramesh (2002). Sjøberg et al. (2005) surveyed ten notable venues in the period from 1993 to 2002. From 5,453 articles, the authors found only 103 experiments (corresponding to about 2%). More recently, Falcao et al. (2015) conducted a similar study surveying six renowned venues in the period 2003 and 2013. The authors observed an increase in experiments in software engineering (6% of analyzed studies). However, the increase is not significantly different from Sjøberg et al. (2005)'s results. Regarding quality of experiment in software engineering, Kitchenham et al. (2013) noticed improvements in last decade. However, Jørgensen et al. (2016) reported that the proportion of statistically significant results in software engineering experiments is much higher than what should be expected given the statistical power of the experiments.

In the previous paragraph, we exposed some issues for experiments in software engineering. In fact, experiments are difficult because they face reality and reality is often unpredictable and more complicated than anticipated (TICHY, 2000). Moreover, many researchers have seen experiments as a risky and challenging study to plan and conduct (BUSE; SADOWSKI; WEIMER, 2011). Some arguments against the use of experiments are that such empirical studies are expensive to run, and may lead to inconclusive or negative results (KO; LATOZA; BURNETT, 2015). In particular to human-oriented experiments, an issue is the fact that humans are the subjects applying several treatments to objects. Jørgensen et al. (2016) suggest research practices to increase the trustworthiness of software engineering experiments. According to the authors, a key to this improvement is to avoid conducting studies with unsatisfactory low statistical power. And, one of the most pragmatic ways to increase the statistical power is by increasing the experiment sample size.

From the start of the 21st century, researchers involved in human-oriented experiments in software engineering became concerned about the methodological standards of such studies (KITCHENHAM et al., 2013). In response to this concern a number of researchers proposed various approaches such as procedures, guidelines, instruments, methodologies, and others (JURISTO; MORENO, 2013; WOHLIN et al., 2012; JEDLITSCHKA; CIOLKOWSKI; PFAHL, 2008; KITCHENHAM et al., 2002; BASILI; SHULL; LANUBILE, 1999; CAMPBELL; STANLEY, 2015; BASILI; SELBY; HUTCHENS, 1986; COOK; CAMPBELL; DAY, 1979; SJØBERG et al., 2002). Among these initiatives are general-purpose platforms to support experiments in software engineering (FREIRE, 2015; ARISHOLM et al., 2002a; TRAVASSOS et al., 2004). Such platform follows the Model-driven engineering (MDE) approach. With this approach, researchers can specify their experiments, while a set of tools can support experiment execution, based on its specification. Such platforms have proved to be a powerful tool for supporting experiment execution. For instance, Wang e Arisholm (2009) conducted an experiment where subjects had to modify legacy code. In this experiment, the subjects used the web-based Simula Experiment Support Environment (SESE) (ARISHOLM et al., 2002a) to download the legacy code and task-descriptions, upload task solutions, and answer questionnaires. The main benefits in using SESE were that researchers employed less effort in managing some operational procedures and collecting some variables.

Despite the general-purpose platform benefits, in certain circumstances, such platform can not provide a complete support to some activities of experiments in software engineering. For instance, Müller e Höfer (2007) carried out an experiment comparing experts and novices when using test-driven development. General-purpose platforms only support part of gathering observed data (time spent in each task and changed files). However, an essential observed variable is the number of executed test methods, and how many of them passed or failed. Besides, general-purpose platforms do not comprise such characteristics. Therefore, the authors had to develop a tool only to collect this data. Tools developed only for an experiment allow researchers to collect fine-grained data from subjects. Besides, such tools lead to a less intrusive experiment, more integrated experiments in user's daily work, as recommended by Wohlin (2013), Sjøberg et al. (2002). On the other hand, such tools require a significant programming effort, and frequently they cannot be reused in similar experiments.

Even using different approaches, the Wang e Arisholm (2009) and Müller e Höfer (2007) experiments have some characteristics in common. Both experiments required to complete software development tasks. Moreover, these tasks had to be performed in a development environment, or Integrated Development Environment (IDE). These experiments share other characteristics such as source codes, libraries, and techniques. Considering all these shared characteristics, we infer that these experiments have similar contexts. According to Brézillon (1999), context is defined **"as a set of relevant conditions and its influences that make a situation understandable and unique"**. However, context characterization in software engineering is still is a challenge (DYBÅ et al., 2012; CARTAXO et al., 2015; CARTAXO, 2014). Cartaxo et al. (2015) argue that it is not possible to unify the context characterization for the entire software engineering. However, it is reasonable for sub-topics in software engineering (CARTAXO, 2014).

In summary, we define our research problem as: "Carrying out an experiment in software engineering is challenging task. One of the most common problems when experimenting in software engineering is to increase the experiment sample size. Many solutions were proposed to mitigate this limitation. However, such solutions can not be used in some experiments, since they did not provide a complete support to them. In particular, experiments in the context of coding tasks (debugging legacy code, testing, etc.) are not completely supported by general solutions. And, in such cases, many researchers have to develop a specific support tool to support their experiments."

1.2 STUDY GOALS AND RESEARCH QUESTIONS

Based on the discussion in the previous section, we define the scope, assumption, and hypothesis of this thesis as follows:

- **Scope** – The scope of this work comprises those experiments when subjects have to perform coding activities (maintenance, test, debugging, etc.). Furthermore, those coding activities should be conducted in development environments;
- **Assumption** – We assume that there are characteristics particular to coding experiment context. Where, such characteristics are intrinsically related to coding task executability (such as artifacts, environment settings, and others.). Besides, such characteristics can be identified, described, and standardized;
- **Hypothesis** – Based on coding experiment context descriptions, our hypothesis is that it is feasible to automatize (entirely or partially) some procedures in such experiments, so that this automation can aid executions of such experiments.

Based on the principles presented before, five research questions were defined:

- RQ1: How do the general-purpose platforms support experiments in software engineering?
- RQ2: What are the characteristics particular to coding experiments?
- RQ3: How can the general-purpose platform approach (MDE approach) be extended to (entirely or partially) automatize some procedures in coding experiments?
- RQ4: What are experiment facets aided by an extended MDE approach?
- RQ5: Does an MDE approach based on coding experiment specification aid the execution of coding experiments?

1.3 CONTRIBUTIONS

By answering our research question, this work makes the following contributions:

- Approach. We introduce the idea of, for a specific experiment context (coding experiment), is possible to (entirely or partially) automatize some procedures when executing such experiments. More precisely, we proposed a solution that synergistically exploits context characterization and experiment environment to help researchers to carry our coding experiments by configuring execution environments and collect observed variables automatically. We implemented this idea to the context of coding experiments. However, our idea can be implemented in different SE topics, for instance, Usability experiment, software management experiments, software process experiments, and others.

- Implementation. We proposed a metamodel and developed two support tools. Our metamodel evidences many relevant coding experiment characteristic (needed artifacts, tasks, and so forth.) required by our solution. Regarding platform tools, the first tool is responsible for specifying coding experiment according to our metamodel. Another tool is an Eclipse plugin that is built on our metamodel. It can automatize some of the procedures in a coding experiment, according to its specification.

- Evaluation. We evaluated our solution by replicating a coding experiment from literature (SANTOS; MENDONÇA; SILVA, 2013). We selected this experiment because: (i) it is an experiment involving coding activities and (ii) it is a well documented experiment [1], (iii) it is not too complicated; then a reader can comfortably experience the benefits in specifying experiments and software development together.

1.4 ACHIEVEMENTS

This research has been developed under the co-supervision of Prof.Maria Teresa Baldassarre (University of Bari) and supervision of Sergio Soares (Federal University of Pernambuco). The results currently achieved can be summarized as follows:

- The Ph.D. theme was proposed and presented in two Ph.D. Symposiums:
  - **Ferreira, W.** (2014, May). Together we are stronger: facilitating the conduction of distributed human-oriented experiments. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (p. 56). ACM.
  - Falessi, D., Codabux, Z., Rong, G., Stamelos, I., **Ferreira, W.**, Wiese, I. S., Tsirakidis, P. (2015). Trends in empirical research: the report on the 2014 Doctoral Symposium on Empirical Software Engineering. ACM SIGSOFT Software Engineering Notes, 40(5), 30-35.

[1] Article about the experiment (SANTOS; MENDONÇA; SILVA, 2013), experiment plan, and website: http://wiki.dcc.ufba.br/LES/FindingGdoClassExperiment2012

- A initial version of our metamodel was published at:
  - **Ferreira, W.**, Baldassarre, M. T., Soares, S., Visaggio, G. (2015, December). Toward a Meta-Ontology for Accurate Ontologies to Specify Domain Specific Experiments in Software Engineering. In Proceedings of the 16th International Conference on Product-Focused Software Process Improvement-Volume 9459 (pp. 455-470). Springer-Verlag New York, Inc.
- The Systematic comparison of general-purpose approaches:
  - **Ferreira, W.**, Baldassarre, M. T., Soares, S., Cartaxo, B., Visaggio, G. (2017, June). A Comparative Study of Model-Driven Approaches For Scoping and Planning Experiments. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (pp. 78-87). ACM.
- The collaboration with a preliminary analysis of human-oriented experiments in software engineering resulted in:
  - Master thesis: Falcão, L. C. T. (2016). Analysis of human-centric software engineering experiments: a systematic mapping study.
  - Falcao, L., **Ferreira, W.**, Borges, A., Nepomuceno, V., Soares, S., Baldassare, M. T. (2015, October). An analysis of software engineering experiments using human subjects. In Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on (pp. 1-4). IEEE.
- The Ph.D research collaborated in an analysis of empirical studies published in renowned Empirical Software Engineering Forums. This research resulted in:
  - Borges, A., **Ferreira, W.**, Barreiros, E., Almeida, A., Fonseca, L., Teixeira, E., Soares, S. (2014, September). Support mechanisms to conduct empirical studies in software engineering. In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (p. 50). ACM.
  - Borges, A., **Ferreira, W.**, Barreiros, E., Almeida, A., Fonseca, L., Teixeira, E., Soares, S. (2015, April). Support mechanisms to conduct empirical studies in software engineering: a systematic mapping study. In Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (p. 22). ACM.
- The Ph.D research collaborated in a research about context characterization in software engineering:

  - Cartaxo, B., Almeida, A., Barreiros, E., Saraiva, J., **Ferreira, W.**, Soares, S. (2015). Mechanisms to characterize context of empirical studies in software engineering. In Experimental Software Engineering Latin American Workshop (ESELAW 2015) (pp. 1-14).

- Preliminary analysis of context in Software Product Lines:

  - **Pires, Waldemar**; Soares, Sergio . Modularity Metrics for Conditional Compilation Software Product Lines. In: Latin American Workshop on Aspect-Oriented Software Development, 2013, Brasilia. Proceedings of V Latin American Workshop on Aspect-Oriented Software Development, 2013. v. 5.

Works under evaluation:

- We submitted a paper about our systematic review of the coding experiments (Chapter 5) to Journal of Information and Software Technology.
- We submitted a paper about our approach to specify coding experiments (Chapter 6) to Journal of Computer Standards and Interfaces.

1.5 ORGANIZATION OF THE THESIS

Besides this introductory chapter, we organized this document as follows:

- Chapter 3 reviews fundamental concepts used throughout this study;
- Chapter 4 presents an systematic comparison of the general-purpose approaches;
- Chapter 5 presents a systematic mapping study to investigate coding experiments in software engineering;
- Chapter 6 presents our metamodel to specify coding experiments;
- Chapter 7 presents the proposed tool support for coding experiments;
- Chapter 8 presents a case study conduced with the aim to demonstrate our solution in a real experiment;
- Chapter 9 presents all related work regarding each chapter;
- Chapter 10 presents the final considerations of this work, and proposes future research based on this work;

2 METHODOLOGY

Art and science have their meeting point in method.

—Edward G. Bulwer-Lytton, *Caxtoniana*

2.1 INTRODUCTION

As it is indicated in the title, this chapter includes the research methodology of the this thesis. In more details, the author outlines the research strategy, the research method, the research approach, the methods of data collection, the research process, the type of data analysis, the ethical considerations and the research limitations.

2.2 RESEARCH METHODOLOGY

We present our research methodology based on the GQM approach (BASILI; ROMBACH, 1988). In Table 1, we define our goal based on the GQM template.

Aiming at this goal, we investigated a set of research questions. In GQM, metrics are associated with every question in order to provide measureable answers. However, it is too simplistic to be applied in our research. As so, instead of metrics, we are answering our questions based on existing research.

Basili et al. (BASILI, 1993) and Wohlin et al. (WOHLIN et al., 2012) classify the scientific researches in four categories:

- Scientific - The world is observed and a model is built based on the observation;
- Engineering - The current solutions are studied and changes are proposed, and then evaluated;
- Empirical - A model is proposed and evaluated through empirical studies;
- Analytical - A formal theory is proposed and then compared with empirical observations.

Table 1 – Research goal in GQM template.

Object of study	Experiments in SE
Purpose	Characterize
Focus	Precision
Stakeholder	Researcher
Context factors	Coding experiments

Wohlin et al. (WOHLIN et al., 2012) also classify the research paradigms as:

- Exploratory research is concerned with studying objects in their natural setting and letting the findings emerge from the observations.
- Explanatory research is mainly concerned with quantifying a relationship or to compare two or more groups with the aim of identifying a cause-effect relationship.

Regarding the approach methods, Hayes (HAYES, 2013) classifies them as:

- Deductive methods use general principles to analyze specific results. Deductive reasoning starts from a general principle or theory, and then collecting data, finally analyzing and interpreting it;
- Inductive methods. Specific observations are used to construct general scientific principles, just the reverse of deductive methods;
- Hypothetico-deductive method starts with hypothesis formulation from generalized principles or theory. This hypothesis is subjected to verification by deduction and comparison with the available data. Then the hypothesis is further tested by a series of step-by-step procedures finally leading to either adoption or rejection of the formulated hypothesis.

Finally, we also present the specific strategy and nature of data analysis to answer each question. All research questions and the specification of their scientific researches are described in Table 2.

Table 2 – Research methodology summary.

RQ1: How do the general-purpose platforms support experiments in software engineering?	
Research method	Scientific
Research paradigms	Exploratory
Approach method	Inductive
Nature analysis	Quantitative and qualitative
Specific method	Systematic literature review

RQ2: What are the characteristics particular to coding experiments?	
Research method	Scientific
Research paradigms	Exploratory
Approach method	Hypothetico-deductive
Nature analysis	Quantitative and qualitative
Specific method	Grounded theory

RQ3: How can the general-purpose platform approach (MDE approach) be extended to (entirely or partially) automatize some procedures in coding experiments?	
Research method	Engineering
Research paradigms	Explanatory
Approach method	Deductive
Nature analysis	Qualitative
Specific method	Systematic comparison

RQ4: What are experiment facets aided by an extended MDE approach?	
Research method	Engineering
Research paradigms	Explanatory
Approach method	Deductive
Nature analysis	Qualitative
Specific method	Applied Science

RQ5: Does an MDE approach based on coding experiment specification aid the execution of coding experiments?	
Research method	Empirical
Research paradigms	Explanatory
Approach method	Hypothetico-deductive
Nature analysis	Quantitative and qualitative
Specific method	Case Study

2.3 RESEARCH CONSTRUCTION

In the previous section, we presented a research methodology for each research question. To archive each research question's answer, we followed a specific research task. In the following, we present each research task conducted:

Step 1: *A systematic mapping study to review primary studies on coding experiments in SE.*

In the middle of 2013, our research group (including this Ph.D. candidate) at UFPE conducted a systematic mapping study to identify support mechanisms for empirical studies in software engineering. The mapping study analyzed papers published at three well-known venues since their first editions. This partial result was an input to this step. More precisely, we extended the mapping study by investigating only specific empirical studies, which correspond to coding experiments. As a result, we provide a comprehensive state-of-the-art of the literature related to coding experiments in software engineering. Our results include:

- Characteristics of coding experiments: Inspired on (SJØBERG et al., 2005; KO; LATOZA; BURNETT, 2015), this result compiles the most common characteristics reported by coding experiment;
- Overview of coding experiments: This result presents meta-information about coding experiments, such as coding experiment growth over the years, most production researchers, and most common topics.

Step 1 answers the second research question, and it took place in first and second years of this research. Details about methodology and results are presented in Chapter 5.

Step 2: *A systematic comparison of general-purpose platforms to support experiments in software engineering.*

We used qualitative analysis methods inspired on grounded theory approach to analyze four general-purpose platforms to support experiments. This study addressed four main topics according to guidelines, namely, Wohlin et al. (2012) and Juristo e Moreno (2013). During this step, we identified:

- Benefits and limitations of the general-purpose platforms;
- Opportunities that can be exploited to provide a better support to coding execution.

Step 2 answers the first research question, and it took place in third year of this research. Details about methodology and results are presented in Chapter 4.

Step 3: *Proposing coding experiment specification.*

Based on first step result, we proposed a solution to specify the coding experiment characteristics. This step aims to define a metamodel containing all relevant information in coding experiment context characterization to provide automatic support for coding experiment procedures. Our metamodel followed the principles:

- Compliant with all general-purpose platform;
- Focus on coding experiment context characteristics that allow certain automation of coding experiment procedures.

Step 3 answers the third research question, and it took place also at the third year of this research. For details about the proposed metamodel, see Chapter 6.

Step 4: Developing tool support for coding experiments.

Based on the metamodel proposed in the last step, this step aims to develop a tool suite to reduce the effort when conducting coding experiments. The current version comprises two tools:

- Modeling tool where researchers can specify coding experiments based on our metamodel. This specification has to be compliant with general-purpose platforms. This compliance is fundamental to allow interactions between solutions. So that, models from other solution may be used as input to our solution and vice-versa;
- Enhanced IDE, our solution is integrated into a popular development tool, Eclipse IDE (ECLIPSE, 2006). It allows coding experiments as close as possible to real development environments. In principle, our solution is valid to other IDE, such as NetBeans and VisualStudio. However, due to limited development resources, we implemented it only to Eclipse.

Step 4 answers the fourth research question, and it took place at the third and fourth years of this research. For more details, see Chapter 7.

Step 5: Assessing and adjusting our solution.

The model and tools were assessed by a coding experiment replication (SANTOS; MENDONÇA; SILVA, 2013). The evaluation took in consideration:

- Capability to support a co-located coding experiment using post-graduate students as subjects. We assess our solution by comparing the effort to carry out the experiment replication against the effort reported by the original study;
- Comments, suggestions, and critics from subjects when using our solution to carry out the replication;

- We compiled the results and made adjustments and improvements on our solution.

Step 5 answers the last research question, and it took place at the fourth and fifth years of this research. Details about this step are presented in Chapter 8.

2.4 CHAPTER SUMMARY

In this chapter, the theoretical and philosophical assumptions underlying the research methodology in the empirical field were reviewed. In addition, a discussion of the research design for this study was made. However, details about each specific method is described in each chapter.

3 BACKGROUND

If I have seen further it is by standing on the shoulders of Giants.

—Isaac Newton

3.1 INTRODUCTION

The demand for empirical studies in Software Engineering (SE) is not new; there is a growing need from the SE community to put it into practice. Basili, Selby e Hutchens (1986) was one of the first studies to explore this area. In the following decades, this study motivated many initiatives to improve, disseminate, and foster empirical strategies in SE. This research is among these initiatives. However, before presenting the results of our research, we have to present a brief but essential background for the reader to understand the context in which this research is inserted. Firstly, Section 3.2 presents a brief overview of empirical strategies in SE. After, we dedicated one section (Section 3.3) only to presents the essential elements of the experiments in SE and experiments in other fields (Section 3.3). Another concept fundamental for this research is context definition, Section 3.4 presents the most common definitions of context in SE. In our work, models are not only documentation artifacts, but they are also central artifacts that allows experiment creation and execution driven by models. So that, according to (SILVA, 2015), the most suitable approach for our context is Model-Driven Engineering (MDE). Section 3.5 presents some basic concepts about MDE. Finally, the summary of this chapter is described in Section 3.6.

3.2 EMPIRICAL STUDIES IN SOFTWARE ENGINEERING

The software engineering research investigates real-world events to foster the development of new mechanisms (such as technologies, processes, methods, techniques, and languages) to support their activities. It aims at improving the quality of software products and it increases the development process productivity (WEYUKER, 2011). Summarily, software engineering research is concerned in investigating how mechanisms work, understand their limits, and propose solutions. In this context, empirical methods provide a consistent method to validate the SE phenomena, generating more accurate evidence and facilitating technology transfer to industry (SJOBERG; DYBA; JORGENSEN, 2007). SE is heavily influenced by human factors (like social and behavioral sciences). Therefore, human factors have a significant impact on software development process and on the quality of the software produced. Moreover, despite the efforts to develop automatic tools and approaches, SE is still dependent on human influence (ROBSON; MCCARTAN, 2016). As a result, SE and the empirical studies have faced several difficulties related to human factors.

As said before, human factors have a fundamental importance in SE. Therefore, the most appropriated empirical methods are those from disciplines related to study human behavior, such as the psychology (regarding the individual level of the human behavior) or the sociology (regarding the team and organizational levels). Methods tailored for these disciplines have known flaws, and can only provide limited, qualified evidence about the phenomena being studied. However, each method is flawed differently, and viable research strategies use multiple methods, applied in a complementary way to address the weaknesses of each method (EASTERBROOK et al., 2008). In the following sections, we present each empirical method adopted in SE.

3.2.1 Experiments

An experiment (or controlled experiment) is an inquiry into a testable hypothesis where one or more variables are managed to measure their effect on one or more dependent variables. A precondition for experimenting is to have clear hypotheses. The hypothesis (and any theory from which it is drawn) drive all steps of the experimental design, including deciding which variables have to be covered and how to measure them.

3.2.2 Case studies

Yin (YIN, 2013) introduces the case study as "an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not evident." Also, case studies offer in-depth understanding of how and why certain phenomena occur. A precondition for conducting a case study is a to have clear research question concerned with how or why certain phenomena occur.

3.2.3 Survey

Survey research is used to identify characteristics of a large population. It is most closely linked with questionnaires for data collection. However, surveys can also use structured interviews, or data logging techniques. The main characteristics of a survey research are (i) the selection of a representative sample of a population, and (ii) the data analysis techniques used to generalize the results from the sample to its population. A precondition for conducting a survey is a to have clear research question that demands information about the nature of a particular target population.

3.2.4 Combination methods

Besides the previously mentioned methods, Easterbrook et al. (2008) cite another approach combining those methods. The choice of each method depends on investigation prerequisites, research purpose, available resources, and how data has to be collected and analyzed. Easterbrook et al. (2008) presents more details about this approach.

Notwithstanding prospects associated with empirical software engineering, studies observed a low empirical evaluation in SE, hindering its progress as a science and delaying adoption of new technologies (TICHY, 1998; JURISTO; VEGAS, 2009; SJØBERG et al., 2005; FALCAO, 2016). Furthermore, Juristo e Moreno (2013) present some arguments against the use of empirical methods: lack of training in those methods on the side of practitioners, difficulties in understanding and analyzing empirical data; empirical studies conducted to check ideas from others are not often published. In this context, Sjøberg et al. (2005) identified a discrepancy between the numbers of experiments and the number of new technologies arising in SE. Wohlin e Aurum (2015) cite other factors that make empirical research in SE particularly challenging. For example, in addition to studying the technology usage, it is also necessary to investigate social and cognitive processes that are associated with human activities. In this context, researchers must have a strong background when choosing methods and guidelines to perform an empirical study (WOHLIN; AURUM, 2015).

3.3 EXPERIMENTS IN SOFTWARE ENGINEERING

Experimentation is a traditional scientific method for identifying cause-effect relationships (BASILI; GREEN, 1994). Experiments (or controlled experiments) are carried out when we want to control a situation and want to manage behavior directly, precisely and systematically (BASKERVILLE; WOOD-HARPER, 1996). When conducting a formal experiment, a researcher wants to study the outcome by varying some of the input variables to a process. There are two kinds of variables in an experiment, independent and dependent variables. Fig. 1 presents an overview of variable relationships in an experiment.

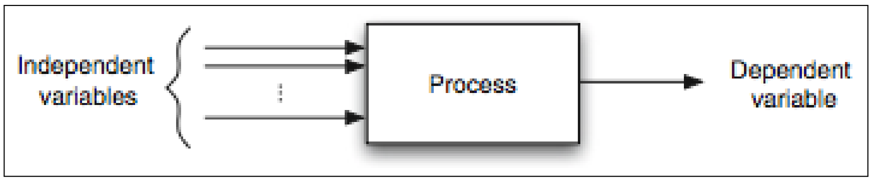


Figure 1 – Illustration of independent and dependent variables (Adapted from Wohlin et al. (2012)).

Any variable that a researcher wants to study its changing effects in other variables is called dependent variable (or response variable). An experiment can have many dependent variables, however, for the sake of simplicity, they usually have only one dependent variable. On the other hand, any variable in the process that is manipulated and controlled is called independent variable.

As said before, an experiment observes the effect of changing one or more independent variables. Those variables are called factors. The other independent variables are controlled at a fixed level throughout the experiment, or else the researcher cannot say if the factor or another variable causes the effect. Treatment is one particular value of a factor. The choice of treatment, and at which values the other independent variable shall have, is part of the experiment design (Fig. 2). The design of experiments is the design of any task that aims to describe or explain the variation of information under conditions that are hypothesized to reflect the variation. The importance of the experiment is not only to get an answer, but also experiment design is about getting right data so a researcher can get an answer that's valid. More details about design of experiment are found in Wohlin et al. (2012).

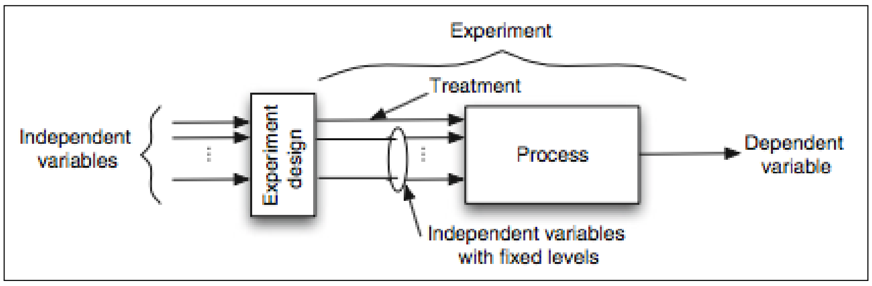


Figure 2 – Illustration of an experiment (Adapted from Wohlin et al. (2012)).

In an experiment, the combination of objects and subjects is called as treatments. An object can, for example, be a source code that shall be reviewed through different inspection techniques. The people that administer the treatment are called subjects. The characteristics of both objects and subjects are independent variables in an experiment.

An experiment consists of a set of tests (sometimes called trials) where each test is a combination of treatment, subject, and object. This type of test should not be confused with statistical tests. Statistical tests verify a hypothesis by statistically checking the observed data. The number of tests affects the experimental error and provides an opportunity to estimate the experimental factor effect. The experimental error helps us to know how much confidence we can place in the experimental results.

According to Wohlin et al. (2012), experiments can be classified as human-oriented or technology-oriented. In human-oriented experiments, humans apply different treatments to objects. For example, two inspection methods are applied to two pieces of code. In technology-oriented experiments, typically different tools are applied to different objects. For example, two test cases are executed on the same program. The human-oriented experiments have less control than the technology-oriented one, since humans behave differently at different occasions, while tools (mostly) are deterministic (WOHLIN et al., 2012).

3.3.1 Experiment Process

In SE, processes offer a set of steps that support activities (for example, software development process). Processes are essential since they can be used as checklists and guidelines of what to do and how to do it. In experiments, several steps have to be taken, and they have to be in a specific order. Thus, a process for how to perform experiments is needed.

Wohlin et al. (2012) proposed a experiment model. This experiment process was formulated to make sure that the proper actions are taken to ensure a successful experiment. It is unfortunately not uncommon that some factor is overlooked before the experiment, and the oversight prevents researchers from doing the planned analysis and hence they are unable to draw valid conclusions. The objective, of having a process, is to provide support in setting up and conducting an experiment.

The starting point for an experiment is the insight. It represents the idea that an experiment would be a possible way of evaluating whatever a researcher is interested in. In other words, we have to realize whether an experiment is appropriate for the question we are going to investigate. As we said before, this is by no means always evident, in particular since empirical studies are not frequently used within computer science and software engineering Sjøberg et al. (2005). Tichy (1998) present some argumentation regarding why computer scientist have to perform more experiments. Assuming that an experiment is appropriate for the research context, then it is essential to plan the experiment carefully to avoid unnecessary mistakes.

The experiment process can be divided into the following main activities.

- Scoping is the first step, where we limit the experiment regarding problem, objective, and goals;
- Planning comes next, where the design of the experiment is determined, the instrumentation is considered, and the threats to the experiment are evaluated;
- Operation of the experiment follows from the design. In the operational activity, subjects apply each treatment and measurements are collected;
- During the analysis, the collected data is dissected and evaluated;
- In presentation and package, the results are presented and packaged.

An overview of the experiment process including the activities, is presented in Fig. 3.

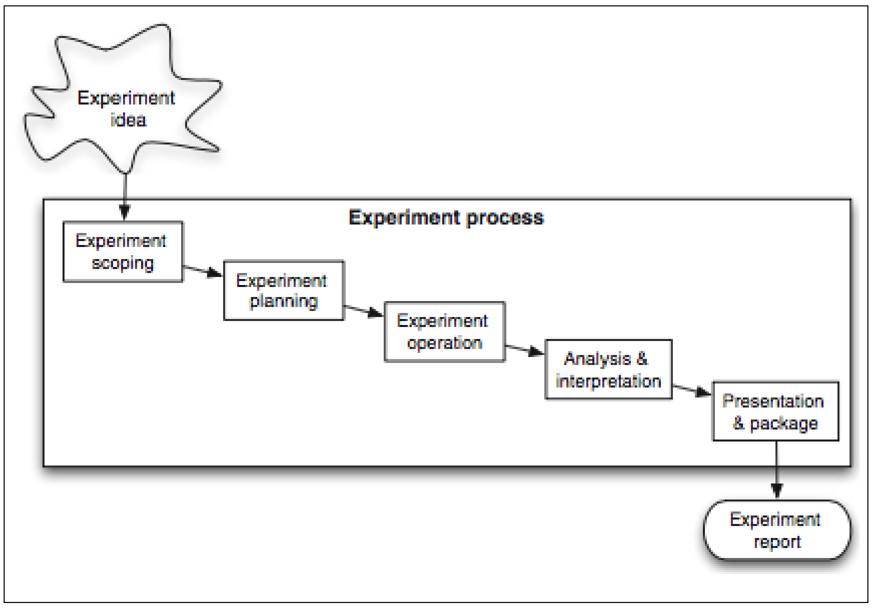


Figure 3 – Overview of the experiment process (Adapted from Wohlin et al. (2012)).

The process is not assumed to be a 'true' waterfall model, therefore an activity must necessarily be finished before starting the next activity. The process is partly iterative, and it may be necessary to go back and refine a previous activity before continuing with the experiment. The main exception is when the operation of the experiment has started; then it is not possible to go back to the scoping and planning of the experiment. In operation, the subjects are influenced by each treatment. Consequently, it is impossible to use the same subjects when returning to the operation phase of the experiment process.

3.4 CONTEXT IN SOFTWARE ENGINEERING

A notion of context usually is used to indicate a phenomenon, event, action or discourse, according to its environment and its consequences (DIJK, 2015). Another definition was presented by Brézillon (1999), in which context is a set of relevant conditions and influences that make a situation unique and understandable. Finally, Dybå et al. (2012) presents other context definition focusing on ubiquitous computing, any information that can be used to characterize a situation of an entity. An entity may be a person, a place, or even a relevant object. Moreover, it may be involved in the interaction between a user and an application, including the user and application in itself.

Given the plurality of perspectives to define context, Dijk (2015) developed an extensive multidisciplinary work that maps the concept of context in humanities, social sciences and even disciplines like artificial intelligence in computer science. Through this study, it is possible to see the context influences when analyzing discourse, literature, art, semiotics, and language in general Dybå et al. (2012). In areas such as sociology and anthropology, which intensively use ethnographic methods, approaches that take into account context are gaining notoriety (HYMES, 2005). An interesting fact emerges when Dijk (2015) shows that there is more work about a context in areas like artificial intelligence and natural language processing, than in psychology. Such approaches represent contexts as models, so that such models are mainly used in areas like pervasive and context-sensitive ubiquitous computing. A more deitailed discussion about this topic is presented by Zimmermann, Lorenz e Oppermann (2007).

Many studies present a need for a correct and formal characterization of context in empirical studies. In the following, we present some articles that make this claim and their arguments:

- While attempting to replicate an experiment, Lung et al. (2008) mention a lack of contextual information as one of the main difficulties encountered. Moreover, there is a need for a good characterization of context when preparing a replication of empirical studies;
- Jedlitschka e Ciolkowski (2004) observed a lack of rigor to report context in empirical studies. This lack makes it difficult technology-transfer, as professionals do not discern whether a result applies to their specific environment;
- Lopes et al. (2010) make it clear that poor context characterization in primary studies creates problems for researchers conducting secondary studies;
- Basili, Shull e Lanubile (1999) mentions the importance of contextual information to create families of controlled experiments.

There are many definitions and interpretations about a context in different knowledge areas. Coupled with the fact that intuitively anything can be potentially relevant to the investigation of a phenomenon, Dijk (2015) argues that theory about context can quickly become a theory of everything. Moreover, Brézillon (1999) states that context is strongly dependent on the domain investigated, and cannot be treated abstractly.

3.5 MODEL-DRIVEN ENGINEERING

In the previous sections, we presented concepts about the problem domain. More precisely, since our work investigates limitations in designing and conducting experiments in SE, therefore we presented the most relevant concepts about experimentation in SE. In this section, we present concepts about the solution domain. In this context, due to researcher's expertise, our solution is based MDE approach. So that, for the sake of uniformity and to void misconceptions, this section introduces essential concepts underlying the MDE approach.

Before presenting any MDE's specific concept, we have to define MDE. Ten years ago, the OMG proposed the Model Driven Architecture (MDA) approach to deal with the separation of platform dependent and independent aspects in information systems (KLEPPE; WARMER; BAST, 2003). Since then, the initial idea of MDA evolved and Model Driven Engineering (MDE) is being increasingly promoted to handle separation and combination of various kinds of concerns in software or data engineering. MDE is more general than the set of standards and practices recommended by the OMG's MDA proposal. In MDE the concept of model designates not only OMG models but a lot of other artifacts like XML documents, Java programs, RDBMS data, etc. Moreover, among these artifacts are included models that can be specified by languages like UML, ER, etc. But we include also the Domain specific languages (DSL).

3.5.1 Model

In the lack of a common definition for "model", in the following, we present some popular attempts to define the concept of model:

1. A model is a collection of statements about the system under study (SEIDEWITZ, 2003);
2. A model is an abstraction of a (real or language-based) system providing predictions or inferences to be made (KÜHNE, 2006);
3. A model is a reduced representation of a system that highlights its properties of interest from a given viewpoint (SELIC, 2003);
4. A model is a simplification of a system to facilitate the answer to questions about the original system (BÉZIVIN; GERBÉ, 2001);
5. A model is a system that helps to define and to give answers of the system under study without the need to consider it directly (SILVA, 2015).

As we can see, there is some consensus that a model describes a system under study. However, it does not mean that a model is the system in itself, with its identity, complexity, elements, relations, and others. However, considering our research's context, we adopted the Silva (2015)'s definition. Regarding that the "system", in our context, is the experiment process to be modeled.

3.5.2 Metamodel

Like for model definition, there is no consensus about metamodel definition. Moreover, some of them are unclear or too limited, for instance, the OMG defines it as "a metamodel is a model of models" (Object Management Group (OMG), 2011). Other authors have thought extensively about this concept. In the following we present the most popular definitions:

- A metamodel is a model that defines the language for expressing models (SOLEY et al., 2000);
- A metamodel is a model for languages of models (FAVRE; NGUYEN, 2005);
- A metamodel is a specification model for which the systems under study can be specified as models in a certain modeling language (SEIDEWITZ, 2003).

We can summarize the metamodel definition as a basement to build any model. Kühne (2006) presents an in-depth analysis of this topic. In the context of this research, a metamodel represents the minimal set of all mandatory context information required to carry out an experiment in SE.

3.5.3 Modeling Language

A modeling language is defined by a metamodel so that it is a set of all viable models that are in conformance with its corresponding metamodel. Silva (2015) provides an accurate definition of modeling language: "modeling language as a set of all possible models that are conformant with the modeling language's abstract syntax, represented by one or more concrete syntax and that satisfy a given semantics. Additionally, the pragmatics (of a modeling language) helps and guides how to use it in the most appropriate way".

The modeling languages adopted to specify models and metamodels in this work are UML (OMG, 2009) and Ecore (STEINBERG et al., 2008). We choose these languages due to an expertise of our research group.

3.5.4 Software Products, Platforms, and Transformations

MDE approach alleges use of modeling languages help to specify models in a reliable level of abstraction, and also those models can be used to support the development process (ATKINSON; KUHNE, 2003; VÖLTER et al., 2013; SELIC, 2008). Silva (2015) go beyond this pleading, they define MDE as "a system composed of a non-trivial integration of software platforms, artifacts directly written by researchers, and eventually models directly executable in the context of a particular software platform". This same definition can be applied to our domain.

In general, software platforms mean an integrated set of computational components which can be used to develop and/or execute a class of software products (BRAMBILLA; CABOT; WIMMER, 2012; VOELTER et al., 2013; GREENFIELD; SHORT, 2003). Usually, each component provides different functionalities through reuse and extensibility mechanisms. As a component, we refer to technologies such as middleware, software libraries, application frameworks, and software components, but also database management systems, web servers, content management and document management systems, and workflow management systems. All this gamut of components may be used to support experiments in SE.

The artifacts are also elements in MDE process. Some artifacts might be only relevant during the design phase while other artifacts might be of interest at execution. Nevertheless, any artifact is tightly dependent on the dedicated platforms. Many examples of these artifacts might be considered like source and binary codes, scripts, and even documentation files, including the models themselves.

Models are a central concept of the MDE approach. On the one hand, a model can be created directly by users (i.e., researchers) or can be produced automatically from transformations and, then, still edited and refined. In our context, models are manually created, however, it may be generated from other models. Nevertheless, it is important to emphasize that, a useful MDE approach, models must be defined consistently and rigorously.

3.6 CHAPTER SUMMARY

In this chapter, we have discussed the essential concepts in the SE area, including evidence based on SE, empirical studies, and primarily controlled experiments. We also presented a discussion about context, and how difficult is to define it. All concepts presented in this chapter are fundamental to understand the thesis' domain problem. On the other hand, all basic concepts about MDE and its terminology and technologies are basements for a full understanding of the solution proposed in this research. Finally, with this knowledge, the reader will be able to understand the context in which this research is inserted.

4 A COMPARATIVE STUDY OF MDE APPROACHES TO SUPPORT EXPERIMENTS

In essence, science is a perpetual search for an intelligent and integrated comprehension of the world we live in.

—Cornelius Bernardus Van Neil

4.1 INTRODUCTION

In software engineering (SE) community, there is an increasing need for experiments to develop or improve SE solutions (JURISTO; MORENO, 2013). Since at least the 80s, this community has been discussing how to provide better support to such empirical studies (BASILI; SELBY; HUTCHENS, 1986). With this goal in mind, many researchers have proposed several approaches, such as guidelines (WOHLIN et al., 2012; JURISTO; MORENO, 2013; BASILI, 1993) and tools (ARISHOLM et al., 2002a; TRAVASSOS et al., 2004). Many promising tools adopted a Model-driven Environment (MDE) approach to support experimentation is SE, assuming that the experimental protocol can be (partly or wholly) considered as a model. We have to bear in mind that the experimental protocol serves as a guideline for establishing information as what is expected from the experiment, what it consists of, how it is going to be executed, the data to be collected, and so on. Therefore, an MDE infrastructure can support an SE experiment based on a specification of its experiment protocol as a model.

An important disclaimer has to be made, models are not the only manner to specify experiments to provide support to them. For instance, experiment protocols in itself are a valuable source of information to prove support to experiments. However, these documents are written in natural language, and natural languages are ambiguous. This ambiguity brings threats to a experiment support that a model based solution (i.e. MDE solution) do not suffer such threats.

Freire et al. (2013) carried out a systematic literature review on automated support for controlled experiments. Considering their results, we identified four MDE approaches that support SE experiments: ESEML (CARTAXO et al., 2012), ExpDSL (FREIRE et al., 2013), Exper Ontology (GARCIA et al., 2008), and eSEE (TRAVASSOS et al., 2004). The first two solutions are based on Domain Specific Languages (DSL) (FOWLER, 2010). Besides, the last two solutions are based on ontologies (CALERO; RUIZ; PIATTINI, 2006).

Each approach above have brought significant benefits in supporting SE experiments. However, our experience in conducting experiments in SE suggests limitations and potential for future improvements in these approaches. Aiming at systematizing our impressions, this chapter presents a systematic analysis of MDE approaches regarding their capabilities to support experimentation in SE. To guide our systematic analysis, we adopted three essential guidelines to conduct experiments in SE (WOHLIN et al., 2012; JURISTO; MORENO, 2013; JEDLITSCHKA; CIOLKOWSKI; PFAHL, 2008). From these guidelines, we extracted eight comparison criteria: (i) standard empirical concepts, (ii) goals and targets, (iii) involved variables, (iv) subject description, (v) design of experiment, (vi) tasks and activities, (vii) instruments and measurements, and (viii) the threats to research validity. Our focus here is on individual experiments from the perspective a researcher. However, our results are relevant to other contexts, like replication, family of experiment, and meta-studies.

The rest of the chapter is composed as follows. Section 4.2 provides an overview of the approaches to specifying SE experiments. Section 4.3 presents the criteria used to analyze and compare the approaches. Section 4.4 presents an analyzes using the criteria previously defined. Section 4.5 discusses the results from our analysis. Section 4.6 concludes and offers suggestions for further research.

4.2 AN OVERVIEW OF EXPERIMENT SPECIFICATION MODELS

As mentioned in Section 4.1, some MDE approaches had emerged to support SE experiment. We identified four models used by such MDE approaches (Table 3). For the sake of clarity, we will present each approach regarding their support to an experiment comparing two programming languages (PRECHELT, 2000). In this experiment, the authors asked to students to develop a same problem in two different programming languages, Java and Python. As results, the authors recorded the time to finish the development.

Table 3 – Models to specify SE experiments.

	Reference	Year	Paradigm
ExpDSL	Freire et al. (2013)	2013	DSL
ESEML	Cartaxo et al. (2012)	2012	DSL
Exper Ontology	Garcia et al. (2008)	2008	Ontology
eSEE	Travassos et al. (2004)	2004	Ontology

- ExpDSL (Experiment Domain-Specific Language) also comprises a DSL and tool. However, besides supporting the specification of the experimental procedure, the tool only supports monitoring the execution of an experiment. The model for specifying experiments in ExpDSL has four views (process, metrics, experimental plan, questionnaire). The process view is responsible for defining the activities, artifacts, and roles. The metric view describes the metrics collected during the experiment execution. The experimental plan view describes information such as the factors or the statistical design. Finally, the questionnaire view represents all surveys to collect quantitative and qualitative data from participants. Many characteristics in Prechelt (2000) can be specified with ExpDSL. Moreover, their tool could manage activities in Prechelt (2000)'s experiment, such as, the observation of the time spent in developing for each language.

- ESEML (Empirical Software Engineering Modeling Language) is a DSL and tool that supports the specification of experimental plans in SE. The authors describe the DSL following a model also called ESEML. According to the authors, the language enables a researcher to represent all relevant information while the tool allows an automated generation of the experimental plan in PDF. With ESEML, Prechelt (2000) could specify part of experiment protocol.

- ExperOntology (Experiment Ontology) is an ontology whose concepts were created to accommodate the representation of SE experiments. It aims to facilitate the reviewing and understanding of experimental lab packages. This ontology is composed of two levels of detail. The first refers to the general concepts of controlled experiments (similar to the experimental plan view of ExpDSL). The second level only focuses on the laboratory package. Related to ESEML and ExpDSL, there is a tool based on this ontology that supports the execution of SE experiments (SCATALON; GARCIA; CORREIA, 2011). Similar to ExpDSL, in the context of Prechelt (2000), ExperOntology tool can collect and configure the experiment environment.

- eSEE (Experimental Software Engineering Environment) is an infrastructure capable of instantiating SE environments to manage knowledge about the definition, planning, execution, and packaging of SE experiments. There are two key components to specify experiments in this infrastructure: the glossary and ontologies. The first aims at establishing a standard terminology in Experimental SE area. The ontologies represent the formalization of the knowledge expressed in the glossary's list of terms. This work focuses on the ontologies. Moreover their tool concerned with supporting scientific workflows are mainly concerned with the execution and analysis phase. In Prechelt (2000), their tool could support the specification of each language and the problems to be solved.

Freire et al. (FREIRE et al., 2013) cited other four tools that support SE experiments (SESE, FIRE, Ginger2, and Mechanical Turk). However, these tools do not follow an MDE approach.

4.3 COMPARISON CRITERIA

Proper criteria are needed to interpret and scrutinize the analysis of the models presented in Section 4.2. We choose eight criteria that are relevant and complementary for addressing our analysis. We extracted them from Wohlin et al. (2012). The first criterion is *general to any empirical study*. Other two criteria concern scoping phase in (WOHLIN et al., 2012): *Goals* and *Involved Variables*. The remaining five criteria concern planning phase: *Subject Description, Design of Experiment, Task, Instrument*, and *Validity Evaluation*. In the following paragraphs, we detail each criterion.

Criterion 1: Basic Concepts. Empirical studies share some standard features in their description and research methodology. This information is an excellent means for introducing the study, no matter what its field and theme (DONOVAN; LAUDAN, 2012). There are some different definitions for most common concepts in empirical research. In this study, we adopted as comparison guideline the APA Manual (ASSOCIATION et al., 1994). The Chapter 2 of this manual describes the minimal information, which should be presented in any empirical study. They are (i) title, (ii) authorship information (such as name, institute affiliation, department, and contact), (iii) abstract. Moreover, according to Jedlitschka, Ciolkowski e Pfahl (2008) other keywords standard features which are fundamental in empirical studies are keywords and bibliography.

Criterion 2: Goal. Primarily, goals define the scope of an experiment. The purpose of its clarity is to ensure that all relevant experiment's aspects were identified before its planning or execution phase. According to Wohlin et al. (2012), and Juristo e Moreno (2013), goal identification is part of the scoping phase. However, this criterion is not only restricted to goal specification. It also includes the research questions and hypothesis. As guideline, we adopted the GQM (Goal-Question-Metric) manual (BASILI; ROMBACH, 1988) and PICO (Population, Intervention, Control, Outcome) frameworks (TREFIL, 2001). The GQM paradigm is a mechanism for defining and evaluating a set of operational goals, using measurement. In this paradigm, the goals are defined in an operational, traceable way by refining them into a set of quantifiable questions that are used to extract the appropriate information. Therefore, solutions have to make a clear relationship among these concepts. For approaches closer to PICO frameworks, goals have to be described as results of the description of Population, Intervention, Control, and Outcomes.

Criterion 3: Involved Variables. When experimenting, we explore outcomes by varying input variables (WOHLIN et al., 2012). Mostly, there are two kinds of variables in experiments, independent and dependent variables. The independent variables are those variables that we can control and change. The independent variable is designed to cause an effect that is measured in the dependent variable. The specification of a variable includes choosing the measurement scales and ranges. Typically, changes in the dependent variables are due to systematic changes in the independent variable, rather than to variations in any uncontrolled extraneous variables (moderator-mediator variables) (BARON; KENNY, 1986). Often there is only one dependent variable, and it should, therefore, be derived directly from the hypothesis. A variable is mostly not directly measurable, and we have to measure it via an indirect measure. The description of the variables is the heart of experiments. Any solution to support experiments has to allow the specification of experiment's variables. However, there is a vast range of possibilities to specify the experiment's variables (WOHLIN et al., 2012). During the analysis, all this information have to be taken into consideration.

Criterion 4: Subject Description. Typically in SE experiments, humans are the subjects applying different treatments to objects. It implies several limitations to control an experiment (WOHLIN et al., 2012; JURISTO; MORENO, 2013). Firstly, humans have different skills and abilities, which in itself may be an independent variable. Secondly, people learn over time, which means that if one subject applies two methods, the order of application of the methods may matter, and also the same object cannot be used for both occasions. Thirdly, human-oriented experiments are impacted by all sorts of influences and threats, due to the subject's ability to guess what the experimenter expects, their motivation for doing the tasks, and others. Hence it is critical for the outcome of the experiment to know how subjects are selected and treated. We have to consider when analyzing the models to describe the subject's characteristics. Moreover, similar to involved variables, there is a vast range of possibilities to specify this information (WOHLIN et al., 2012).

Criterion 5: Design of Experiment. A key factor of any experiment is the Design of Experiment (DoE). The statistical analysis is applied depending on the chosen design, and the measurement scales. The designs range from a simple experiment with a single factor to complex experiments with many factors. We adopted as references to analyze the solutions two guidelines Montgomery (2008) and Juristo e Moreno (2013). The minimal set of DoE is Completely Randomized, Randomized Complete Block; Factorial Design 2x2, and Factorial Design 2k. However, other DoE also can be supported (such as the Taguchi method (PEACE, 1993)).

Criterion 6: Task. In general, an experiment task involves a change in a human response while interacting with a computer (WOHLIN et al., 2012). A task is any activity in an experiment that is being studied. The qualities of a useful task are representative and discriminateive (ZIGURS; BUCKLAND, 1998; JURISTO; MORENO, 2013). Representative tasks mean that subjects perform a task most closely to reality. It improves external validity (but it also may compromise internal validity). Besides, a task has to discriminate among observed variables. It increases the likelihood of a statistically significant outcome (i.e., the sought-after "change" occurs). Most experiment tasks are performance-based (e.g., time, accuracy, and so forth) or skill-based (e.g., inserting an equation, programming a destination location, or others). However, sometimes the task is knowledge-based (e.g., "Use an Internet search interface to find the birth date of Chico Buarque"). In this case, subjects become contaminated (in a sense) after the first run of a task, since they have acquired the knowledge. In this context, such information has to be taken into considerations when designing an experiment. Therefore, an MDE approach to support experiments in SE has to define it precisely.

Criterion 7: Instrument. There are three types of instruments in SE experiments, namely objects, guidelines, and measurement instruments (WOHLIN et al., 2012). Experiment objects may be, for example, specification or code documents. Guidelines are needed to guide participants during the experiment. Guidelines include, for example, process descriptions, and checklists. Finally, measurement instruments are any artifices used in an experiment to collect data. An MDE approach to support experiments in SE has to specify all instruments.

Criterion 8: Validity Evaluation. An important question concerning experiment results is how valid are such results. It is essential to consider experiment validity already in the planning phase to design an adequate validity of the experiment results. In principle, adequate validity refers to the fact that results are valid for the population of interest. First of all, the results should be accurate for the population from which the sample as drawn. Secondly, it may be of interest to generalize the results to a broader population. Results are said adequate regarding the validity if they are valid for the population to which the researcher would like to generalize. There are four types of threats to validity: Internal, external, conclusion, and construct (WOHLIN et al., 2012; JURISTO; MORENO, 2013). An MDE approach for SE experiments has to specify all threats involved in the experiment. Besides, it has to allow the specification of what actions were taken to mitigate each risk. It is valid not only to MDE approaches, but to any other approach.

4.4 COMPARATIVE ANALYSIS

In this section, the MDE approaches presented in Section 4.2 are analyzed using the criteria defined in Section 4.3. Each criterion is analyzed separately, namely Basic Concepts in Section 4.4.1, Goals in Section 4.4.2, Involved Variables in Section 4.4.3, Subject Description in Section 4.4.4, Design of Experiment (DoE) in Section 4.4.5, Task in Section 4.4.6, Instruments in Section 4.4.7, and Validity Evaluation in Section 4.4.8.

4.4.1 Basic Elements

There are many possibilities to satisfy the *basic elements* criteria presented in Section 4.3. Table 4 summarizes the accuracy of each solution in representing it.

In ESEML, there is an absence of such entities to determine the basic elements. Possibly, this solution focuses essentially on experiment concepts. In the other hand, the

Table 4 – Basic Element Comparison.

	Basic Elements
ExpDSL	title, abstract, authors, and keywords
ESEML	-
Exper Ontology	author
eSEE	title, abstract, authors, keywords, and references

ExpDSL has an entire view only to describe the experimental plan. In this view, a researcher specifies almost all basic elements presented in Section 4.3 (Table 4). However, this DSL allows only the specification of authors name (it does not include other information like the author's affiliation and institution). The ExperOntology has two entities to describe the researchers involved in an experiment (the Designer and the Replicator). The Designer creates the original experiment, while the Replicators have their profiles associated with the replicated experiment.

Only the eSEE allows the specification of all the concept presented in Section 4.3. The entity *Document of Experimental Study* in the sub-ontology *Experiment Package* can specify all basic elements. Besides, this ontology specifies some meta-information about the experiment protocol: comments, date of last change, date of creation, idiom, file name, links title, and version.

4.4.2 Objective

During the scoping phase, researchers determine fundamental information about their experiments. Table 5 summarizes how each solution specifies experiment objectives.

Table 5 – Goal Specification Comparison.

	Objective Definition	**Frameworks**
ExpDSL	Goal, Question, and Hypothesis	GQM
ESEML	Goal, Question, and Hypothesis	GQM
Exper Ontology	Goal and Hypothesis	-
eSEE	Goal, Question, and Hypothesis	GQM and PICO

As presented in Table 5, the ExpDSL stisfy completly the criterion of experiment goal specification. In *Experimental Plan View*, there are three sub-packages: *Goal*, *Research Question*, and *Research Hypothesis*. Regarding the *Goal* package, there are two relevant entities, the *Simple Goal* or *Structured Goal*. The first specifies the goal in a paragraph in natural language. The *Structured Goal* defines the goal following the GQM template. The entities *Research Question* and *Research Hypothesis* are similar to *Simple Goal*. They define their information in a paragraph in natural language. Besides, both *Research Question* and *Research Hypothesis* are associated to a goal.

In ESEML, the goal specification follows only the GQM template. The goals are refined into a set of quantifiable questions so that goals become operational and traceable. Moreover, the questions define a particular set of metrics and provide a framework for their interpretation. Besides, ESEML has the entity *Hypothesis* to specify each hypothesis in natural language. However, there is no relationship between hypotheses and a question, goal, or other elements (such as variables or instruments).

The ExperOntology has a sub-Ontology (*Lab Packages Ontology*) only to specify the experiment objective. In this package, two classes represent the experiment hypothesis: the *Initial Hypothesis* and the *Formalized Hypothesis*. The *Initial Hypothesis* is a preliminary hypothesis proposed while the experiment scope is being explored. It comprises the objects of study in agreement with the *Purpose* and *Context*. After, an *Initial Hypothesis* generates a *Formalized Hypothesis*. This last entity includes the *Null Hypothesis* and *Alternative Hypothesis*. Unfortunately, the ExperOntology does not provide an explicit support to define questions and goals. However, the entity *Purpose* can be interpreted as a goal.

Finally, the eSEE has two sub-ontologies for defining goals, questions, and hypothesis. The sub-ontology of *Scientific Research* specifies the experiment goals, which each goal follows the GQM template. However, the eSEE does not make a clear relationship between goals, questions, and metrics (as recommended by the GQM paradigm). On the other hand, only this ontology specifies all elements required to specify research goal as recommended by PICO. The sub-ontology of *Controlled Study* defines hypothesis (both the null and alternative) and research questions. Finally, this ontology connects each hypothesis to its goals and questions.

4.4.3 Involved Variables

An experiment can prevent its data analysis from masking effects of experimental manipulations upon the objects under study. Therefore, a precise description of such objects and involved variables is fundamental. All MDE solutions specify the involved variables. However, they differ in specification approach.

In *Experimental Plan View*, the ExpDSL proposes three entities for defining the experiment variables. The *DepVariable* describes each dependent variable. Each *DepVariable* includes at least one research question or hypothesis. This association is important to allow data traceability. The *Parameter* is an entity used to characterize the experiment context, which means that each trial results will be specific to their conditions defined by these parameters. An experiment has a set of *Parameter* where each parameter has a fixed value. Both *Parameter* and *DepVariable* has a *scaleType* and *range*. The last entity is the *Factor*, defining the experiment's factors.

ESEML has five entities to describe variables involved in an experiment. The *Response Variables* entity outlines the experiment outputs. The *Factor* describes the features that intentionally vary during the experiment executions. Moreover, the *Treatment* identifies one particular value of a *Factor*. The entity *Parameter* specifies fixed characteristics at a given value in the experiment, so they should not vary throughout the experiment execution process. Finally, the last entity is the *Experimental Object*. It represents the objects that "suffers" the experiment performance. All these entities have to be in the natural language.

In ExperOntology, the *Ontology for Lab Package* describes the involved variables. Two entities represent the *dependent* and *independent variables* with homonym names. Both *dependent* and *independent variables* have an association with a *Formalized Hypothesis* (Section 4.4.2). This ontology identifies the factor by associating an *Independent Variable* to the *Experimental Design* (Section 4.4.5). Besides, an *Independent Variable* can be linked to *Experiment Object*. This association identifies the controlled variables as source codes or involved tools. Finally, a *Dependent Variable* can have an association with the analysis specification. This relationship specifies how observed data shall be analyzed.

The *Sub-Ontology of Quantitative Method* defines all the involved variables in eSEE. As the previously presented models, the eSEE represents dependent and independent variables with an entity with homonym names. This ontology also defines *Block Variables* and *Not-controlled variables*. Besides, it defines how to manipulate all these variables and their scales. Finally, the results of each dependent variable are associated with its corresponding hypothesis.

4.4.4 Subject Description

The selection of participants is essential when experimenting. It represents the sample from a population. The experiment results can only be generalized to the population if the selection is representative of the desired population. Therefore, the participant information has to be in its experiment specification. Moreover, as said in Section 4.3, each MDE solution has a proper manner to specify the participant characteristics.

The ESEML has an entity to specify the participant characteristics, *Subject*. This object describes the individuals who apply, the used techniques, and the methods in an experimental unit. However, it not clear which participant characteristic were covered. Besides, this entity is not associated with any other entity in the model.

Unfortunately, the ExpDSL does not allow to describe the experiment sample. However, MDE tool supports such information. In the tool, a researcher can register each participant of the experiment (ALEIXO et al., 2010).

In ExperOntology, the entities *Subject* and *Profile* are responsible to describe the subject characteristics. The *Profile* records each important characteristic of subject background. By taking subject background, a researcher can identify possible influence on experiment results. Moreover, the *Profile* has a link to the class *Questionnaire* that specifies the instrument to collect such data (details in Section 4.4.4). Furthermore, *Subject* has an association with three other classes: *Experimental Plan* (details in Section 4.4.5), *Execution Plan* (details in Section 4.4.6), and *Results*.

The eSEE adopts a different approach to describe the subjects involved in an SE experiments. Two sub-ontologies specify such information: *Sub-ontology of Participants, roles, and responsibilities* and *Sub-ontology of the target study population*. The first ontology defines experiment executors (professional researcher, student, and professional developer), visitor, and a software engineer (or a researcher). This ontology also specifies roles for each person involved in the experiment (such as average participant, replicator, or a practitioner). The former sub-ontology determines population sampling strategies. It follows the sampling strategies presented by Wohlin et al. (2012).

4.4.5 Design of Experiment (DoE)

Researchers apply statistical analysis methods on the collected data to interpret them so that they may draw meaningful conclusions. Such method must be carefully planned and designed to get the most out of the experiment. It is also important to know which statistical analyses we can apply depending on the chosen design, and the used measurement scales. Therefore, MDE solutions for SE experiments have to specify precisely Desing of Experiment (DoE). Table 6 summarizes how each solution specifies this information. The first column presents the standard DoE supported by each solution. The second column shows if there are associations between DoE entities and other entities (factors, variables, question, and forth).

Table 6 – DoE Specification Comparison.

	Design of Experiments	**Associated with other entities**
ExpDSL	Completely Randomized Design, Randomized Complete Block Design, Latin Square, and Others	Yes
ESEML	One-Sample Comparison, Two-Sample Comparison, Latin Square, and Others	No
Exper Ontology	General Description	Yes
eSEE	Completely Randomized Design; Randomized Complete Block Design; Factorial Design 2x2; and Factorial Design 2k	Yes

The entity *Design* in ESEML specifies experiment designs. This entity defines the following standard DoE: Latin square, one sample comparison, and two-sample comparison. The ESEML proponents are not clear how *Design* entity is associated with other entities in the model.

In ExpDSL, the element *DoE* is responsible for defining the statistical design of the experiment. Currently, the model supports three statistical models: Completely Randomized Design, Randomized Complete Block Design, or Latin Square. When a researcher needs to set any other DoE, he/she should select the entity "Other". This entity has a text field to detail the DoE specificities.

The ExperOntology has *Experimental Design* entity to specify the standard DoEs. It is built combining experiment objects, independent variables, and subjects, in conjunction with their corresponding hypothesis. The authors do not make it clear if this ontology supports all standard DoEs.

In eSEE, The sub-ontology *Design Process* is responsible for describing the Doe. In principle, this sub-ontology supports four standard DoEs: Completely Randomized Design, Randomized Complete Block Design, Factorial Design 2x2, and Factorial Design 2k. Besides, this sub-ontology can define DoE basic principles (Balanced, Blocking, or Randomized) depending on sampling characteristics

4.4.6 Task

A precise task description is a key component in an experimental design. When describing each action to be performed in the task, it is fundamental to describe the required artifacts (including instruments - Section 4.4.7), variables to observe (Section 4.4.3), and task order. Table 7 summarizes task representation analysis for each MDE approach.

Table 7 – Task Specification Comparison.

	Association		
	Artifacts	Variables	Task Order
ExpDSL	Yes	Yes	Sequential
ESEML	No	No	Not Clear
Exper Ontology	Yes	Yes	Sequential
eSEE	Yes	Yes	Sequential

In ESEML, the *Design* entity specifies information about tasks. According to their authors, the DoE, task description, and task order are closely related. Therefore, the *Design* entity summarizes all this information. Moreover, this information is described in natural language. A limitation in ESEML is that a task is not associated with any other element in the model.

The ExpDSL provides extensive support to specify experiment tasks. The *Task* entity has an association with other three entities: *Artifact*, *Field*, and *Questionnaire*. The first defines all artifacts required to execute each task. The entities *Field* and *Questionnaire* are somehow similar; they demand explicitly some information provided by each participant (details in Section 4.4.4). However, there is no particular entity for specifying task order. On the other hand, the association among tasks defines experiment task order. The experiment ends when there are no more tasks associated.

The ExperOntology defines relevant information about the task execution in *Execution Plan* entity. An *Execution Plan* has an association with *Planned Task* and the *Performed Task*. These objects specify each task status whether a task was accomplished or not. Regarding task order, the sequential association between the *Execution Plan* and the *Task* specifies the sequential order. Finally, the *Experiment Object* defines each artifact involved in each task. It can be a *Technology* (Technique, method, or tool) or an *Artifact*.

In eSEE, the *Sub-ontology of the controlled study* is responsible for defining experimental tasks. Regarding task order, eSEE adopts a model similar to ExpDSL; each task knows its next tasks. Moreover, eSEE has a complete description of all artifacts involved in the experiment (details in Section 4.4.7). However, it is not clear how each artifact can be associated with a particular task.

4.4.7 Instruments

There are three types instruments in an experiment, specifically objects, guidelines, and measurement instruments. The objects (or experiment objects) may be, for example, blueprint, source code, and documents. Guidelines are needed to conduct the participants in the experiment. The measurement instruments are any artifice used in an experiment to collect data. Table 8 summarizes the representation in each model for each type of instrument.

Table 8 – Instrument Specification Comparison.

	Objects	**Guidelines**	**Instruments**
ExpDSL	Partially	No	No
ESEML	Yes	Partially	Partially
Exper Ontology	Partilly	Yes	Yes
eSEE	Yes	Yes	Yes

The *Object* entity specifies a set of instruments in ESEML. As said before, the *Object* defines the experimental unit. However, the concept of experimental unit sometimes does not represent an instrument. For instance, the experimental unit in an SE experiment can then be the software project as a whole. Regarding other types of instruments (guidelines and questionnaires), the ESEML is not clear about whether it can or cannot be specified.

In ExpDSL, the *Artifact* entity is responsible for determining instruments. According to the authors, an artifact is one of many kinds of tangible by-products produced during the development of software. There is not a clear distinction between an object or a guideline since an artifact can be either a piece of source code as a PDF with the task description. Besides, considering that each task has a description, we can interpret it also as guidelines for a task.

Similar to ExpDSL, the ExperOntolotogy also define the instruments through an entity called *Artifacts*. It is an abstract entity, it is realized by the following entities: *Document*, *Questionnaire*, *Form*, or *Tool*. The *Document* can describe the guidelines. Furthermore, the *Forms* or *Questionnaires* can describe some measurement instruments. Moreover, the generic *Artifacts* can specify the other artifacts (such as source code or unit tests).

The eSEE has a whole sub-ontology to describe the experiment instruments. In this sub-ontology, the*Document* entity stores relevant information such as lists, questionnaires, and observation notes. This class can represent both measurement instruments and guidelines. Another entity in this sub-ontology is the *Software Artifacts*. This class represents any software document such as user's documents, requirements documents, analysis documents, design documents, system documentation, business domain model, software process model, and data test case. Finally, the *Code Document* represents the program text document, function, and interface.

4.4.8 Validity Evaluation

A fundamental question concerning experiment results is how valid are these results. It is important to consider the question of validity already in the planning phase to plan for an adequate validity of the experiment results. As said in Section 4.3, we analyzed the MDE solutions in the light of four threats to validity: construct, conclusion, internal, and external validity. Table 9 summarizes how each solution specifies experiment validity.

Table 9 – Validity Specification Comparison.

	Type of Treats	**Mitigation Actions**
ExpDSL	All	No
ESEML	Internal, and External	Yes
Exper Ontology	All	No
eSEE	All	No

In ESEML, the *Validity* entity specifies risks in an experiment. However, this entity covers only two types of validity, internal and external validity. Moreover, the authors are not clear about whether there is a way for specifying actions to mitigate these threats.

Similar to ESEML, ExpDSL has also only one entity to define threats, *Threat to Validity*. Each *Theat To Validity* has an identifier, description, type of threat (Conclusion, Internal, Construct, External), and optionally a control action (in control action attribute). The control action is described in natural language.

ExperOntology defines one abstract type to specify a general threat (*Threat to Validity*) and four subtypes to determine a particular type of validity. According to the authors, the conclusion validity refers to the relationship between the treatment and outcome; internal validity refers to the points that assure there is a causal relationship between the factors and the outcome; the construct validity concerns with the relation between theory and observation; and the external validity concerns with generalization. As ESEML, the ExperOntology is not clear about how to document actions to mitigate threats.

Finally, the eSEE defines the threat to validity in the *Sub-ontology of Experiment Planning*. The experimental plan entity has four attributes, one for each type of threat. However, similar to ESEML and ExperOntology, the authors do not mention how to specify actions to mitigate risks.

4.5 DISCUSSION

In this section, we discuss the results of our analysis. Our purpose is to identify some improvements for each MDE solution. Besides, we also suggest in which scenarios each solution is most suitable. Table 10 summarizes our analysis.

Table 10 – Comparative analysis summary.

	Basic	Goals	Variables	Subjects	DoE	Task	Instrument	Validity
eSEE	✓	✓	✓	✓	✓	∼	✓	✓
Exper Ontology	∼	✓	✓	✓	✓	∼	∼	✓
ESEML	✗	✓	✓	✓	∼	∼	∼	✓
ExpDSL	∼	✓	✓	✗	✓	∼	∼	✓

✓: It specifies all relevant characteristics
∼: It specifies some relevant characteristics
✗: It does not specify the relevant characteristics

Basic Concepts. Different MDE solutions focus on the different aspects of basics concepts. Firstly, we have to raise the question whether it is more profitable to cover more and to be more extensive, or cover less and to be more precise. In our analysis, we realized that 'completeness' is a concept that must be associated both with vertical (i.e., the level of detail) and horizontal (i.e., core elements) dimensions. None of the models evaluated were both extensive and precise. There is always a trade-off in that sense. With that said, eSEE is the most adequate to specify an experiment since it is the most extensive. However, it is too complex. A complete specification of an experiment in eSEE is a challenging work, and even with support tools. Besides, it has few spots to provide flexibility in the specification. Regarding flexibility, ExpDSL is recommended. For instance, it allows a researcher to specify abstracts as structured abstract or straightforward abstract.

Goals. As recommended by Wohlin et al. (2012), almost all MDE solutions provide support to specify goals. Only ExperOntogy does not provide complete support for it.

Similar to the Basic Concepts, the most extensive solution is the eSEE. However, the specification of an experiment is not a straightforward process. As recommended by Wohlin et al. (2012), it is an incremental and iterative process. With that said, only the ExperOntology takes care of documenting this process. All other solutions focus only on hypothesis' final version. Regarding flexibility, ExpDSL allows a researcher to specify goals following two methods, a simple specification or by GQM template.

Involved Variables. All MDE solutions describe this concept. It was expected since the variable description is a fundamental concept in any experiment. However, there is no consensus about a terminology about his concept in SE experiments. For instance, all models agree with the term Dependent Variable. However, the ESEML and ExpDSL use the terms Parameter, while the ExperOntology use the terms independent variable for the same concept [1]. The ExperOntology and eSEE specify the distinction between factors and parameters. Regarding the format to specify the involved variables, the ESEML and ExperOntology specify it in natural language. The ExpDSL allows the description in scales and ranges. However, this DSL makes some confusion between what is the variable description and its measurement description[2]. The eSEE emphasizes this distinction between variable description and measurement. Variables and Measurements are specified separately. Moreover, the eSEE also allows an association between these two concepts.

Subject Description. Similar to Variables, subject characterization is a core concept in any experiment. Surprisingly, the ExpDSL does not provide support to specify it. Both ESEML and ExperOntology support a limited specification of the experiment sample. Only the eSEE provides a complete framework to specify almost all relevant characteristics of the experiment sample.

Design of Experiment (DoE). All models provide excellent support to specify the standard DoEs. However, only the eSEE supports design principles depending on the population sampling. Besides, only the ExpDSL is flexible so that it supports uncommon DoEs. For instance, the ExpDSL can specify a DoE following the Taguchi methods (PEACE, 1993).

Tasks. The ESEML is not clear about task execution. This model defines tasks, trials, and DoE altogether (Section 4.4.5). The ExpDSL and eSEE specify the sequence of tasks by each task referencing its next tasks. In ExperOntology, an entity has an ordered association to identify the task order. Unfortunately, all these models fall in the same limitation, only specify sequential execution is allowed. Wohlin et al. (2012) argue that a random task order is desirable since it can avoid the learning effect. An example of experiments that did not follow a sequential implementation of tasks (SANTOS; MENDONÇA; SILVA, 2013).

Validity Evaluation. All solutions allow the specification of threats to validity. Some

[1] According to Wohlin et al. (2012), factor and parameters are independent variables

[2] Wohlin et al. (2012) discuss this distinction in Chapter 7.

models specify it by one entity represent one type of threat (ExperOntology and eSEE). Other models specify it as an attribute in the generic threat. Moreover, only the ExpDSL can specify the actions to mitigate the threats.

Previously, we presented an analysis each approach regarding our selected criterion separately. Below, we present a more general summary:

- **ESEML**. This model is the most simple way of specifying the experiment. Few entities specify any experiment. Models in ESEML are used to generate an initial version of the experimental protocol. However, it has to be enriched with more precise information. We recommend this approach for simple or initial experiments (proof of concept) performed by experienced researcher;

- **ExpDSL**. This approach is more expressible them the ESEML. The authors focused on a more practical perspective of the experiment. So that, the authors also provide a tool that supports certain automation of some experiment management tasks (FREIRE et al., 2013). Regarding extensibility, it falls in the same limitation as for the ESEML, any extension of the model has to be mapped in changes in their specific language (Ecore (STEINBERG et al., 2008)). This approach is recommended to researcher that wants exploit the automation and management of experiment tasks provided by their tool support;

- ExperOntology. This ontology provides (a full or partial) support to all criteria evaluated in this work. Unlike the previous approaches, this model can be easily extended, since there is no bound language. So that, any information not specified in ExperOntology may be specified in other languages. We recommend ExperOntology when the researcher wants traceability in some design decisions for the experiment. This model has entities to specify historical data and information. For instance, this model has entities like *Initial Hypothesis* and *Final Hypothesis*, and *Planned Task* and *Executed Task*;

- **eSEE**. It is the most complete and expressive model. This model fulfill almost all evaluated criteria (only arbitrary task execution cannot be specified and mitigation tasks). Moreover, similar to ExpOntology, any extension of this model no need updates in any bound language. However, any extension of this model is not a simple task, since a change in one class may provoke a cascade change in many other entities. We recommend such model for an inexperienced researcher, or for researchers that want a precise experimental specification.

4.6 CHAPTER SUMMARY

This chapter has presented an analytical study of the currently available MDE solutions to support experiments in SE. We have introduced a set of perspectives based on fundamental elements of main guidelines for SE experiments. The selected criteria were (i) basic concepts, (ii) goals, (iii) involved variables, (iv) subject description, (v) Design of Experiment (DoE), (vi) tasks, (vii) instruments, and (viii) threats to research validity. Other criteria could be used, such as, effort or time spent to perform an experiment. Such criteria are interesting for future work.

We have identified several common deficiencies in these solutions, especially in task description and instruments. Moreover, we identified the limitations and benefits of each model. In summary, we can say that the most complete and expressive model is the eSEE. On the other hand, the most simple approach is the ESEML. Besides, the ExpDSL focus on the most a most practical perspective of experiment execution. Finally, we recommend ExperOntology when the researchers want some tractability in their design decisions.

With the acknowledgement of strengths and limitations of each approach, we identify what approach is most suitable to coding experiment context. Moreover, we can integrate these approaches to provide a better support to coding experiments.

5 AN ANALYSIS OF CODING EXPERIMENTS IN SOFTWARE ENGINEERING

Whenever truth stands in the mind unaccompanied by the evidence upon which it depends, it cannot properly be said to be apprehended at all.
—William Godwin

5.1 INTRODUCTION

Aiming at understanding and assisting empirical studies in SE, our research group (including the thesis's author) conducted a systematic mapping of the world literature on empirical studies in SE (BORGES et al., 2015). More precisely, the venues covered were Evaluation and Assessment in Software Engineering Conference (EASE), Empirical Software Engineering and Measurement (ESEM), and Empirical Software Engineering Journal (ESEJ). They were mapped since their first editions. As result of this research, we identified several mechanisms used to support empirical studies in SE. Moreover, we observed an increased number of empirical studies over the years. However, a considerable quantity of such studies did not report employing any specific empirical method (16% of analyzed studies), and others did not report using any mechanism to guide their inquiries (53% of total).

The main limitation of the previously mentioned research was data heterogeneity. From a methodological perspective, diverse empirical methods report different data. On the one hand, experiments and case studies share several concepts such as objectives, research questions, and collected data. On the other hand, experiments focus on reporting information like variables, control factors, etc. While case studies report case selection, cases and units analysis, etc. In studies such as meta-analysis, the greater the study's data heterogeneity severity and the lower study's reliability (KITCHENHAM, 2004). The same is valid for other research methods (ethnography, action research, etc.) (EASTERBROOK et al., 2008).

With the objective of overcoming the above limitation, our research group (including the thesis's author) carried out another research to refine and extend our first results. By refining, we mean selecting from our previous dataset only those empirical studies that performed experiments which humans were subjects (human-oriented experiments) (FALCAO, 2016). Moreover, we only included papers published in the period between 2003 and 2013. Other venues also were included, namely, Information and Software Technology Journal (IST), and IEEE Transactions on Software Engineering (TSE). This new research allowed us to analyse more accurately human-oriented experiments in SE. Moreover, we compared our results with a similar survey conducted ten years ago (SJØBERG et al., 2005).

As a tangential conclusion from the research designed earlier, we observed that experiments in SE are not as similar as we expected. Cartaxo et al. (2015) argue that it is not possible to unify a precise context characterization for all experiments in SE. To illustrate this issue, we can take as example two experiments, one on HCI (Human-computer Interface) (BENBASAT; DEXTER; MASULIS, 1981) and other that evaluates two coding techniques (VOKÁČ et al., 2004). While the previous experiment focuses on the user interactions, and then its relevant variables are mouse clicks and screen records, in the latter experiment, relevant variables are changed lines of code and time spent to develop them.

Considering all limitations of our works mentioned earlier, we decided to investigate a reduced set of experiments in SE. This set comprises those experiments with activities involving coding, the scopus of this thesis. According to (IEEE Computer Society, 2014, Chapter 6), coding activities can vary from developing an entirely new system to updating a legacy system with new functionalities. In the last decades, many solutions (techniques, methods, and tools) have been proposed to make coding activities less resource consuming or to improve the overall code quality. In this scenario, experiments appear as a remarkable strategy to provide an empirical basis for trustful comparisons among the solutions.

According to Wohlin et al. (2012), any experiment in SE follows a process divided into five phases: Scoping, Planning, Operation, Analysis & Interpretation, and Presentation & Package. In this research, we focus only on Planning and Operation.

In summary, the purpose of this chapter is (1) to provide an overview of coding experiment characteristics and (2) to distinguish the general experiment characteristics and the coding experiment characteristics. We believe that an outline of the coding experiments state-of-the-art provides to SE community a better understanding of the current status of coding experiments. Furthermore, it identifies new perspectives and gaps that foster the development of mechanisms to aid such kind of experiment.

The remainder of the chapter is structured as follows: Section 5.2 presents our definition of coding experiments. Section 5.3 explains the used research method. The results of mapping of systematic maps in coding experiments are presented in Section 5.4. Section 5.5 discusses some findings of our research and Section 5.6 show a brief summary.

5.2 CODING EXPERIMENTS

There is no consensus about a definition of "experiment" in SE community. There are slightly different definitions of experiments in SE, with the most appropriate depending on the context and problem of interest (WOHLIN et al., 2012; JURISTO; MORENO, 2013). In this study, we adopted the same operational definition used by Sjøberg et al. (2005) to avoid misunderstandings. We believe this definition is appropriated to the context of this research.

> "Controlled experiment in software engineering (operational definition): A randomized experiment or a quasi-experiment in which individuals or teams (the experimental units) conduct one or more software engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages, or tools (the treatments)."

As we see from the quote above, the adopted definition explicitly refers to tasks performed by individuals or teams. Some authors classify such experiments as a human-centric experiment (or human-oriented experiments) opposing to the technology-oriented experiments. In the first, human beings apply the treatments to the experimental objects (WOHLIN et al., 2012; KITCHENHAM et al., 2013). In the technology-oriented experiments, typically different tools are applied to different objects, for example, two test case generation tools are applied to the same program. Kieburtz et al. (1996) is a case of technology-oriented experiments.

5.2.1 Definition

As said before, our focus here is not on generic experiments, but on a smaller set of experiments, the coding experiments. In such experiments, participants have to perform coding activities. According to IEEE Computer Society (2014), coding or programming activities comprise designing, writing, testing, debugging, or maintaining. Code designing activities correspond to conception or invention of a scheme for turning a customer requirement for computer software into the operational software. When performing such activities, developers link application' requirements to coding artifacts. Code writing activities is the actual coding of the design into an appropriate programming language. In code testing activities, developers have to verify if the written code does what it was designed to do. In code debugging activities, developers have to find and fix bugs (faults) in source codes (or design) artifacts. In code maintenance activities, developers have to update, correct, or enhance existing applications.

Usually, experiments with design activities only produce mechanisms to bridge a link between requirements and source codes. Therefore, frequently participants do not have to produce or change any source code. Then, due to this nature, we do not include them in the scope of this work. On the other hand, we consider code inspection and comprehension as coding activities. Code inspection refers to a code peer-review performed by trained individuals, who looks for defects using a well-defined process. Besides, code comprehension (or application comprehension) corresponds to those activities which developers have to perform a set of actions depending on their code understanding.

5.3 METHOD

Following several systematic mapping studies guidelines (PETERSEN; VAKKALANKA; KUZNIARZ, 2015; KEELE, 2007; PETERSEN et al., 2008), we undertook this study in distinct stages: searching for relevant studies, identifying inclusion and exclusion criteria, critical appraisal, data extraction, and synthesis. In the following, we present the search protocol, which has been developed by the thesis' author and later it was reviewed by his supervisors.

5.3.1 Data Sources and Search Strategy

One of the main difficulties in conducting systematic mapping studies is the effort searching relevant research in literature (PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Aiming at reducing this effort, we reused existing databases from previous studies, as introduced in Section 5.1. In the following, we present each adopted database and how they were reused.

Initially, our research group [1] performed a systematic mapping study to identify support mechanisms (process, tool, guideline, and so forth) used to plan and conduct empirical studies in SE community. It included all full papers published at EASE, ESEM, and ESEJ since their first editions. We analyzed 1,098 articles, among primary, secondary, and tertiary studies. We found 362 support mechanisms. A contribution of this systematic mapping study was to provide a catalog of support mechanisms available to SE community interested in empirical research. With this catalog, it is possible to know which resources are being used as a reference to plan and support different kinds of empirical studies and in which contexts they are applied. The complete catalog is available online[2]. The initial results from this systematic mapping study were published in previous studies in ESEM 2014 (BORGES et al., 2014) and EASE 2015 (BORGES et al., 2015). The author collaborated with searching, selecting, and analyzing part of the papers. More details can be found in Borges et al. (2014).

The work mentioned above served as the baseline for another systematic mapping study. In this new research, our goal was to characterize and analyze only human-oriented experiments in SE. We included studies published at ESEJ, ESEM, EASE, ICSE, JSS, and TSE from 2003 to 2013. As result of this mapping study, we found 216 human-oriented experiments. We analyzed methods and procedures used by such experimenters. Moreover, we compared these results with Sjøberg et al. (2005) to see if there have been any changes or recent trends. Similar to the previous work, the Ph.D candidate only collaborated in the review process. The results of work contributes marginally to this

[1] https://sites.google.com/site/eseportal/
[2] https://goo.gl/nOj4Tu

Ph.D. research, however it cannot be considered as part of this research. A more detailed analysis is reported in Falcao et al. (2015), Falcao (2016).

A preeminent critic to our previous work was the observation period (2003 to 2013). Some argued that it is outdated. Therefore, we made an extra effort to extend it to 2016. Moreover, we included another venue in our study, Information and Software Technology Journal (IST). We carried out a "Two-Step" method inspired on Kitchenham et al. (2013) to conduct such extension:

- Automatic search. Unlike Kitchenham et al. (2013), we searched the ScienceDirect database instead of the SCOPUS. In SCOPUS, we could not limit the search by venue. Besides, we used the search string "experiment", while Kitchenham et al. (2013) adopted a complex string. In fact, Dieste e Padua (2007) recommend complex string. However, two years later the same authors refined the research recommending only the string "experiment" (DIESTE; GRIMÁN; JURISTO, 2009). In this stage, we identified 50 papers as possible human-oriented experiments in IST, and more 25 papers published during 2014, 2015, and 2016;
- Manual search. We performed a test-retest validity check (GEWANDTER et al., 2014). Unlike Kitchenham et al. (2013), due to resource limitations, we did not inspect all venue's issues. We examined them by sampling. We selected randomly half of the studies in each issue. After, we classified each of them. We did not found disagreements between our results in this step and the previous step.

After finishing the previous steps, we identified 291 human-oriented experiments published from 2003 to 2016. Finally, the effort in extending our previous research was almost entirely performed by the Ph.D. candidate. Another researcher only collaborated in pair reviewing and quality evaluating.

5.3.2 Inclusion and Exclusion Process

After identifying 291 human-oriented experiments, we had to distinguish which of them correspond to coding experiments according to the adopted definition (Section 5.2.1). Unfortunately, the process to determine if a study is a coding experiment is not straightforward. For instance, in Accioly, Borba e Bonifacio (2012) this information is "Method" section. However, in Santos, Mendonça e Silva (2013), this information can appear in other sections. Therefore, a process of including or excluding articles based only on titles and abstracts was not possible. The adopted process followed a divide and conquered strategy. The 291 human-oriented experiments were divided into two subsets (145 and 146 papers, respectively). One researcher was responsible for reading each paper and include only those papers in agreement with our coding experiment definition. One of the researchers was the Ph.D. candidate, the other researcher was Larissa Falcão, Ph.D. student at Center of Informatics in Federal University of Pernambuco.

A process based on a single author to include or exclude articles poses a threat to the reliability of our mapping study, as we also discuss at threats to validity section (Section 5.5.1). Aiming at threat mitigation (KEELE, 2007), 86 papers were randomly selected from each subset, and reanalyzed by another researcher. Table 11 provides data used in our kappa analysis (WOOD, 2007). The raw data provided by each reviewer is available on-line[3].

Table 11 – Kappa statistic description.

		Reviewer A		Total
		Accepted	Rejected	
Reviewer B	Accepted	43	38	81
	Rejected	4	131	135
Total		47	169	216

As presented in Table 11, we observed 174 agreements (43 acceptances and 131 rejections, 80.56% of papers). Moreover, the number of deals expected by chance is 123.3 (57.06% of the observations). Therefore, we obtain a Kappa value of 0.547, and a standard error equals to 0.058. In a 95% confidence interval (0.434 to 0.661), the strength of agreement is considered to be "moderate". Even with an average Kappa value, we find this value low. Therefore, we revisited papers in disagreement. The majority of differences were experiments involving design tasks using DSLs instead of models (like UML, ER, and so forth). Some authors discuss the distinction between DSL and programming languages (FOWLER, 2010). Summarily, a DSL is a specialized computer language designed for a specific task. On the other hand, general-purpose languages (like C, Python, and Haskell) are designed to write programs with any logic. We focused only on general-purpose languages. Therefore, we agreed in excluding these papers. Finally, we identified 99 articles as possible coding experiments.

5.3.3 Quality Assessment

We selected nine criteria to assess the 99 studies identified by the previous stage. These criteria were adapted from Kitchenham et al. (2013). Table 12 presents the quality evaluation questionnaire.

[3] http://bit.ly/1pXNiD7

Table 12 – Quality Questionnaire adapted from Kitchenham et al. (2013)

#	Question
Category: Questions on Aims	
1	Do the authors clearly state the aims of the research?
Category: Questions on Design, Data Collection, and Data Analysis	
2	Do the authors describe the sample and experimental units?
3	Do the authors describe the design of the experiment?
4	Do the authors describe the data collection procedure, and define the measures?
5	Do the authors define the data analysis procedures?
6	Do the authors discuss potential experimenter bias?
7	Do the authors discuss the limitations of their studies?
Category: Question on Study Outcome	
8	Do the authors state the findings clearly?
9	Is there evidence that other researchers/practitioner can use the experiment setup?
Score: 0=Not at all; 1=Somewhat; 2=Mostly; 3=Fully;	

The measure of study quality was computed by summing the score on all items (total score varies from 0 to 27). Similar to the last phase, the same two researchers measured the quality of each study. On average, the 99 coding experiments obtained a score of 21.3. According to Kitchenham et al. (2013), high-quality works have a score better than 15. This high average score reflects the quality of selected venues. We excluded only two papers due to its low score (score lower than 15). All raw data and questionnaires are available on-line[4]. A secondary result of this phase was to identify 25 papers misclassified as coding experiments. These papers performed experiments involving inspections on design artifacts. Then, they are out of our scope (Section 5.2.1). Finally, we identified 72 papers reporting coding experiments. Among these papers, four papers reported two experiments and other two papers reported three experiments. Therefore, we have in total 80 reported coding experiments.

[4] Due to limitations in Google Spreadsheet™, we had to split our result into two spreadsheets: http://bit.ly/1XMrqW3 and http://bit.ly/21GSZB3

5.3.4 Data Extraction

We developed a data extraction instrument to obtain all relevant data from coding experiments. Initially, our instrument was inspired on Sjøberg et al. (2005) and Falcao et al. (2015). However, during the data extraction process, this spreadsheet evolved. This practice is recommended by Keele (2007) because during the selection process the researchers get involved in the problem domain. The final version of our instrument is available online[5]. The extraction was performed by the Ph.D. candidate and reviewed by Larissa Falcão. Having double-check process in data extraction is a common practice in systematic reviews for social science (PETTICREW; ROBERTS, 2008).

5.3.5 Analysis and classification

The information for each item extracted was tabulated and analyzed. The derived strategies were grouped and given a theme by the first author during analysis. Table 13 presents eight identified themes and their sub-themes. The theme and sub-theme classification was based on Petersen e Ali (2011).

Table 13 – Themes and Sub-themes identified in Coding Experiments.

	Theme	Sub-Themes	Domain
1	Guidelines		Planning and Execution
2	Foundation	Factor, Treatment, Goal, and Hypothesis	Planning
3	Subject	Quantity, Category, Recruitment, and Sampling	Planning and Execution
4	Task	Category, Duration, Order, and Artifact	Planning and Execution
5	Environment		Execution
6	Data Collection	Time, Resources, and Survey	Execution
7	Replication		Planning
8	Threats to Validity		Planning

5.4 RESULTS

5.4.1 Overview of Studies

Figure 4 shows the number of coding experiments identified within the years 2003-2016. We notice a constant interest in coding experiments during the period. It has a smooth decay during 2006-2010, and a significant increase after 2010. A potential reason for this interest in coding experiments may be the evaluation of emerging solutions like TDD and Agile Methods.

[5] available at http://bit.ly/2diQhEw

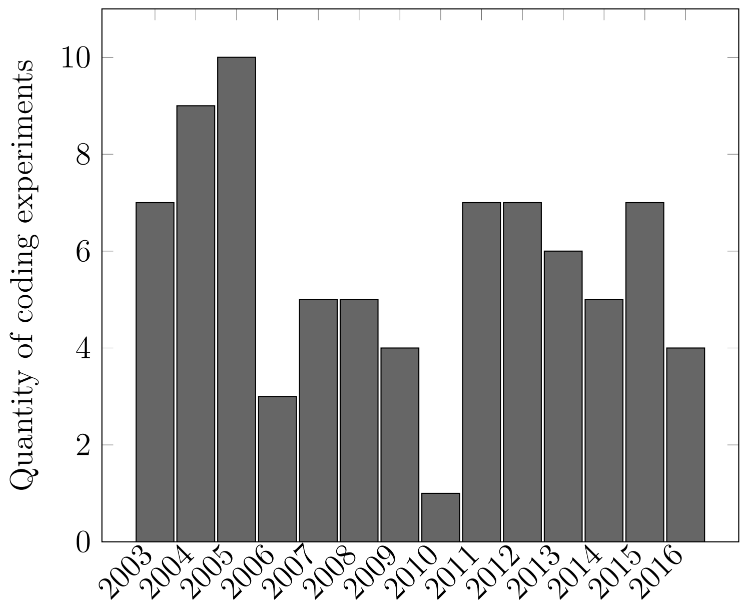


Figure 4 – Distribution of publications over the years.

In this study, peer-reviewed venues (including journals, as well as peer-reviewed conferences) were considered. Figure 5 provides an overview of articles distribution between these sites. A similar number of articles was published in international conferences and scientific journals. Overall, this indicates coding experiments are regarded as valuable scientific contributions, given they are widely published in high-quality forums.

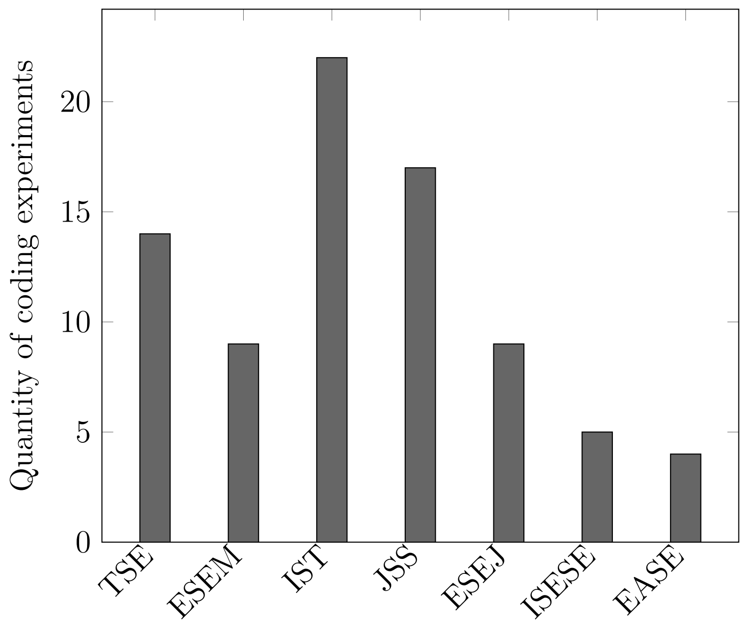


Figure 5 – Distribution of publications by venues.

In total, 198 researchers authored at least one paper about coding experiments. Table 14 presents the four researchers that authored more then three papers. Besides these authors, other six authors published three coding experiment, four published twice, and the remaining authors published once.

Table 14 – Productive Authors

Ranking	Researchers	published papers
1	Dag i. k. Sjøberg	5
	Erik Arisholm	
2	Massimiliano di Penta	4
	Matthias m. Muller	

5.4.2 Guidelines

This section presents some results about the first theme at Table 13, Adopted Guidelines. As stated by Costa (2015), experiments conducted under guidelines tend to have a higher quality. Table 15 presents the ten most frequently cited guidelines. The remaining 11 guidelines were mentioned only once.

Table 15 – Most cited guidelines.

	Guidelines	#
1	Wohlin, Claes, et al. Experimentation in software engineering. Springer Science & Business Media, 2012.	13
2	Juristo, Natalia, and Ana M. Moreno. Basics of software engineering experimentation. Springer Science & Business Media, 2013.	7
3	Basili, Victor R., Forrest Shull, and Filippo Lanubile. Building knowledge through families of experiments. IEEE Transactions on Software Engineering, 1999.	6
	Kitchenham, Barbara A., et al. Preliminary guidelines for empirical research in software engineering. IEEE Transactions on software engineering, 2002.	
5	Basili, et al. Experimentation in software engineering. 1986	2
	Jedlitschka et al. Reporting guidelines for controlled experiments in software engineering 2008	
	Cook and Campbell. Quasi-experimentation: design & analysis issues for field settings. 1979	
	Basili. The goal question metric approach. Encyclopedia of software engineering. 1994	

As we see in Table 15, the most commonly followed guidelines on experimentation are Wohlin et al. (2012), Juristo e Moreno (2013), Basili, Shull e Lanubile (1999), Kitchenham et al. (2002). Our results corroborate with Borges et al. (2015). Besides, other guidelines on experimentation sub-topics also were cited (family of experiments and reporting) (JEDLITSCHKA; CIOLKOWSKI; PFAHL, 2008; KITCHENHAM et al., 2008; BASILI; SHULL; LANU-

BILE, 1999). Moreover, guidelines from outside SE have been also used (ASSOCIATION et al., 2001; THE..., 2002; LINDSAY; EHRENBERG, 1993). The only guidelines not directly related to experimentation were GQM technique guidelines (BASILI; ROMBACH, 1988; BASILI, 1992; SOLINGEN et al., 2002).

It is noteworthy to observe that often many guidelines were combined to support some coding experiments. On the other hand, more than half of coding experiments (46 experiments) had not cited any guidance document. Thus, specific guidelines do not appear to be complete enough to characterize a whole coding experiment. Moreover, we had not found a guideline specific to coding experiments.

5.4.3 Foundations for Experimentation

The second emerging theme from coding experiments was classified as foundations for experimentation. When proposing any experimental design, some key components are factors and treatments (WOHLIN et al., 2012). Table 16 shows the distribution of factor quantity in coding experiments. Our findings indicate that the majority of coding experiments deal with one or two factors. According to Jørgensen et al. (2015), simple experiment design is desirable, since such studies usually achieve more reliable results. The coding experiment with the higher quantity of factors was Ng et al. (2012). This experiment involved six factors and each factor with two treatments. To reduce its experiment complexity, the authors adopted the following strategy: each participant performed only a single coding activity for each factor. Besides, all activities changed only one medium-sized program.

Table 16 – Factor Quantity.

Factors	# of papers
One Factor	66
Two Factors	8
Three Factors	5
Six Factors	1
Total	80

Not only the number of factors impacts the experiment process. The number of treatments by factors also affects it. Table 17 shows the distribution of treatments in coding experiments. Jørgensen et al. (2015) claim that the number of treatments is a way to make an experiment complex. We identified two studies with four treatments by factor Stärk, Prechelt e Jolevski (2012) and Karahasanović, Levine e Thomas (2007). In both cases, the authors mentioned their actions to reduce experiment complexity. Stärk, Prechelt e Jolevski (2012) reduced its complexity by reducing the experiment control, each treatment was a programming language, and each participant only used one of them. Moreover, a small sample was recruited (16 participants). Therefore, a strong statistical analysis was not possible. Karahasanović, Levine e Thomas (2007) took a similar approach. However, this study had a bigger sample (45 participants).

Table 17 – Treatments by Factors.

Treatments by Factor	# of papers
Two Treatments	82
Three Treatments	10
Four Treatments	8
Total	103

Another element that affects the experiment design is the number of hypotheses. Similar to factors and treatments, Jørgensen et al. (2015) also recommend investigating few hypotheses. Table 18 shows the number of studied hypothesis in coding experiments. As we can see, about 70% of coding experiments (52 experiments) examined up to four hypotheses. Moreover, few variables are involved in each hypothesis. For instance, Huang e Holcombe (2009) evaluated the number of tests produced by a set of students and the time to write these tests. In the same experiment, there was a hypothesis about productivity, which is the relation between the number of tests and time to produce them.

Table 18 – Hypothesis Frequency.

Quantity of Hypothesis	# of papers
Two Hypothesis	24
One Hypotheses	10
Three Hypotheses	11
Four Hypothesis	7
Six Hypothesis	11
Ten Hypothesis	4
Five Hypothesis	2
Seven Hypothesis	1
Nine Hypothesis	1
Total	71

The study presented in this chapter found only one study without any reference to factors or treatments. Regarding hypothesis, this mapping identified nine experiments not mentioning their hypothesis. In one hand, these findings indicate that few coding experiments did not report fundamental information. On the other hand, only the existence of such works suggests there is a need for more effort in improving reports.

Considering format issues, we did not observe any pattern when reporting factors and treatments. All identified experiments indicate this information in article's body as plain text. The majority of papers also state their hypothesis as sentences in a paragraph (about 75% of the papers). A guideline to experiments in science recommends the hypothesis format as "if –, then – will happen" (STEFFE; THOMPSON, 2000). However, some coding experiment defines their hypothesis as expressions (HUANG; HOLCOMBE, 2009; CHOI; DEEK; IM, 2008; STURM; KRAMER, 2014; FUCCI et al., 2015; LUCIA et al., 2014). Such format is suggested by other guidelines (WOHLIN et al., 2012; JURISTO; MORENO, 2013).

5.4.4 Subjects

This section presents our findings regarding subjects (Table 13). In theory, notions about subjects are part of the topic foundations for experimentation (Section 5.4.3). However, considering the diversity of information collected, we decided to create a theme only for this topic.

5.4.4.1 Quantity and Categories of Subjects in Coding Experiments

In total, 3807 subjects were involved in the 80 coding experiments. From these 80, 56 were conducted only with students and 16 only with professionals. The reported subject types are divided into categories presented by Table 19. The table also presents the same classification used by Sjøberg et al. (2005) (general experiments in SE published between 1993 and 2003) and Falcao et al. (2015) (general experiments in SE published between 2003 and 2013).

Table 19 – Subject Category.

			Falcao et al. (2015)		Sjøberg et al. (2005)	
Subject Category	N	%	N	%	N	%
Undergraduates	1387	26.25	5768	36.1	2969	54.1
Graduates	1434	30	2703	16.9	594	10.8
Students, type unknown	181	3.75	1023	6.4	1203	21.9
Under and graduates	-	-	2719	17	-	-
Professionals	800	20	2531	15.8	517	9.4
Profess. and Students	-	-	1194	7.4	-	-
Scientists	5	0.13	-	-	74	1.3
Unknown	0	0	5	0	131	2.3
Total	3807	100	15943	100	5488	100

A conclusion draws from Table 20 is an increasing proportion of experiments involving professionals rather than students. We observed about a double amount of experiments involving professionals over Sjøberg et al. (2005)'s results. This increase is partially explained when comparing Falcao et al. (2015) and Sjøberg et al. (2005). However, our results are 5% higher than Falcao et al. (2015). An explanation for this result is the fact that frequently coding experiments assess techniques that are valuable to practitioners (HUANG; HOLCOMBE, 2009; MÜLLER; HÖFER, 2007; CHOI; DEEK; IM, 2008; PHONGPAIBUL; BOEHM, 2007). We also observe an increase in the proportion of coding experiments with graduate students when comparing with undergraduate students. Feitelson (2015) argue that graduate students have a similar profile as practitioners.

Table 20 – Subject Categories meta-information.

		Experiments		Subjects				
Category of subjects		N	%	Mean	Min	Median	Max	Sum
Student only	Undergraduate	21	26.25	50.19	18	38	159	1054
	Graduate	24	30	40.63	4	33	130	975
	Undergraduates and graduates	8	10	87.13	12	96	144	697
	Student (Unknown)	3	3.75	17.16	10	18	25	53
		56	70	49.63	4	38	159	2779
Professional Only		16	20	40.13	5	27.5	196	642
Mixed group of subjects		8	10	48.25	10	32.5	158	386
Total		10	100	47.59	4	31	196	3807

Table 20 presents some details about the sample size of coding experiments. In literature, there are several methods for determining the sample size. Williams, Onsman e Brown (2010) consider five respondents per variable to be analyzed as the lower limit, but the most acceptable way of determination is using a 10:1 ratio (10 samples for one variable). In similar veins, Schreiber et al. (2006) also suggested that each parameter should have at least 10 participants. However, there is not a priori fixed minimum sample size (DYBÅ; KAMPENES; SJØBERG, 2006).

The experiments involving only students are the majority of the coding experiments (70.%). It is similar to Sjøberg et al. (2005)'s results (72.6%). With respect to statistical analysis of sample size, we also observed similarities Sjøberg et al. (2005)'s results, in particular, mean (56) and median (36). Such results indicate a pattern for sample size in experiments. However, some authors suggest attention in the adequacy of sample sizes in experimental SE research to ensure acceptable power levels (DYBÅ; KAMPENES; SJØBERG, 2006; JØRGENSEN et al., 2015).

About the proportion of coding experiments involving professionals, we also observed some similarities with Sjøberg et al. (2005) results. Both studies observed about 20% of total experiments involved only professionals. However, concerning statistical analysis, our results diverge. We observed a higher mean and median. This result indicates more attention is given when performing coding experiments with professionals. The same can be said when analyzing our results about those experiments with students and practitioners.

The number of participants in coding experiment ranges from four to 196, with a mean value of 47.6. Students participated in 64 (80.0%) experiments, either alone or together with professionals. Professionals took part in 24 experiments, either alone or jointly with students(30.0%). These numbers are similar to those found by Sjøberg et al. (2005), where professionals were involved in 24% of the total. On the other hand, Ko, LaToza e Burnett (2015) observed a clear result. In this study, 23% of the experiments involved only students, 56% recruited software professionals. The remaining 23% had not provided any detail about participants. We offer some explanations for these differences: (1) slightly different scope; tool evaluation is near to industry (details in Section 9.3.1) and (2) a broader definition of experiments, Ko, LaToza e Burnett (2015) included case studies and correlational studies as experiments.

There is a considerable literature about differences between students and professionals in SE experiments (FEITELSON, 2015). We found nine coding experiments with both students and practitioners. Only two of them measured the difference in performance between these two groups. In one of them, authors compared pair development and software inspection (PHONGPAIBUL; BOEHM, 2006), and they observed differences regarding quality and effort between students and practitioners. In another experiment, the authors compared experts and novices regarding their adherence to the test-driven development (TDD) processes (MÜLLER; HÖFER, 2007). The authors only observed differences when subjects were inexperienced in TDD.

Still, on the topic of students vs. professionals, we highlight a paper (RUNESON, 2003). In fact, the authors used only graduated and first-year students as subjects in this coding experiment. However, the authors compared their data and the data from a similar experiment with professionals (HAYES; OVER, 1997). The conclusions drawn from the study can neither reject nor accept the hypothesis if there is a clear difference among first-year students, graduate students, and industry people. However, the authors observed a significant difference between first-year students and graduate students. Their data was not sufficient to evaluate similarities or differences between industry people and graduate students. A more in-depth analysis of all these coding experiments is required to draw any precise conclusion about this topic.

We agree with both Sjøberg et al. (2005), and Ko, LaToza e Burnett (2015) that using students in coding experiments should not be avoided, but it should not be restricted to students only studies. In particular, we believe students represent a good sample for initial hypothesis, or for programming learning purposes. Besides, graduate students in Masters or Ph.D. program may also be a representative sample representing junior professionals. However, the precise role of student vs. professional is not defined, which is a topic for further research.

Finally, all coding experiments reported the number of participants (Sjøberg et al. (2005) observed the same). 19 experiments reported subject mortality (dropouts), it is about 24.% of the total. These 19 experiments reported 5.6% mortality, on average. These numbers are also similar to those presented by Sjøberg et al. (2005). Besides, three out of 19 papers explicitly said that no participant abandoned their experiments. However, most articles did not report this information.

5.4.4.2 Information about Subjects

Making conclusions about a much wider population than the experiment sample need some description of its sample (WOHLIN et al., 2012). Such description is any information about various characteristics and their variation (both in the sample or the target population). Sjøberg et al. (2005) observed there is no standard set of information to describe experiment samples. Regarding only coding experiments, we observed some standard in such information. However, the level of detail reported by coding experiments varies substantially. Some articles describe in details their sample characteristics with descriptive statistics (STÄRK; PRECHELT; JOLEVSKI, 2012). Other studies describe the sample characteristics in a paragraph.

In 2015, Siegmund e Schumann (2015) presented a catalog of 39 confounding parameters for program comprehension experiments based on a literature survey. This catalog together with Wohlin et al. (2012) provides a precise compendium to describe proper information about subjects involved in coding experiments. Next, we present the most common sample parameters when describing students or professionals in coding experiments.

In the coding experiment with students, the most regular information about the sample is the programming experience (11 experiments) and experience with Java (seven experiments). Usually, it is measured in years, and students have on average 5.60 years of programming experience and 3.25 years of Java experience. Some coding experiments also involved other programming languages, such as C (two experiments) and PHP (one experiment). Besides, four experiments with graduate student asked about any professional experience. Other cited parameters were: age (nine experiments), gender (six experiments), and course semester (seven experiments). Regarding experiments involved graduate students, just two experiments provided details about subjects' postgraduate degree (Ph.D. or MSc. Students).

For the 24 experiments with professionals, 12 experiments specified the nationality of their samples. The majority of the experiments took place in Europe (the Spain, Norway, Sweden, and the UK). Moreover, some other countries were cited: USA, Brazil, Thailand, India, and Korea. Similar to the experiments with students, five experiments with professionals described the programming experience (on average, four years), and six experiments cited experience with Java. The experience in industry was an issue in five experiments (on average, four years).

An alarming finding was that 30 coding experiments did not provide any information about their sample characteristics. Sample characterization is critical when interpreting experiment's results. Such information is fundamental to studies based on the statistical analysis combining results of multiple scientific studies, such as meta-analysis (KITCHENHAM et al., 2002). In areas like social sciences and biology, standards to characterize the sample are based on regulations and previous studies (SCHULZ et al., 2010). There are some initiatives in adapting such techniques to SE context. For instance, Siegmund et al. (2014) present an approach to measure experience in SE. Besides, for undergraduate students, Feigenspan et al. (2012) found that a self-estimation of language experience on a scale from 1 to 10 correlates moderately with performance on programming tasks. Moreover, Host, Wohlin e Thelin (2005) proposed a classification for the experience of subjects in experiments involving inspection and reading techniques.

5.4.4.3 Recruitment of Subjects

In general, a barrier to experiments in SE is recruiting participants (BUSE; SADOWSKI; WEIMER, 2011; KO; LATOZA; BURNETT, 2015). In coding experiments, it is not different. Next, we present some information about how subjects are being recruited and rewarded in coding experiments.

Table 21 summarizes the data regarding subjects' rewarding. Some experiments rewarded professionals and students distinctly, so there are more reward occurrences in Table 21 (83 reward occurrences) than coding experiments (80 experiments). Comparing our data with Sjøberg et al. (2005)'s results, we notice in both works that many experiments do not provide any information about rewards. Such works either do not reward their subjects, or they do not cite this information. However, regarding the most common reward, we observed *payment*, while Sjøberg et al. (2005) observed *grade and extra credits*. We attribute this difference to the higher number of professionals in coding experiments.

Table 21 – Subject Reward Data

Reward	Number of experiments
Paid	11
Mandatory	3
Grade	2
Competition	3
Extra credit	2
No Reward	21
Unknown	41
Total	83

The most common strategy to recruit subjects is inviting attendees of a course (60 coding experiments). It is especially prevalent when recruiting students. However, we also found some professionals being recruited in training and certification courses. Apart from experiments during courses, 12 experiments were performed *in vivo*. In other words, they were conducted in a real environment. However, in eight experiments, the authors did not describe any recruiting strategy.

The participation was mandatory in four experiments with attendees at a course (undergraduate courses). In two of them, their involvement determined their grades. Besides, in other two experiments subjects had an incentive by earning extra credits for the course. In two experiments, subjects were paid to take part of the experiment. Moreover, the payment usually is cash or a gift card. In three cases, subjects competed for a prize.

Usually, experiments with professionals were part of regular projects or training programs. We classified such experiments as *No Reward* (seven cases). In other six experiments, the authors explicitly said that professionals were paid (an extra in their salaries). Only in one experiment, professionals competed for a prize.

Performing power analysis and sample size estimation is an important aspect when recruiting subjects. It is essential because, without these calculations, the sample size may be too high or low (WOHLIN et al., 2012). If the sample size is too low, the experiment will lack the precision to provide reliable answers to the questions under investigation. If the sample size is too large, time and resources will be wasted, often for minimal gain. Only 16 experiments reported power analysis or sample size estimation (20%). Moreover, only two experiments claimed to adopt random subject sampling. However, none described how the random sampling was carried out. The dominant approach was convenience sampling.

In 2013, Kitchenham et al. (2013) observed an improvement in the quality of SE experiments. However, Jørgensen et al. (2015) demonstrated an unbiased situation did not match their observed proportion of statistically significant tests in SE experiments. Our observations corroborate with Jørgensen et al. (2015) conclusions. Besides, Jørgensen et al. (2015) proposed a list of research practices to support SE experiments with hypothesis testing. We emphasize the importance of such practices also in coding experiments.

5.4.5 Tasks

In experiments, subjects have to allocate their efforts to diverse tasks according to the experimental plan. In coding experiments, a task varies from building a software application to implementing new features into an existing application. This section reports theme and sub-themes related to tasks in coding experiments.

5.4.5.1 Task Categorization

We categorized coding tasks according to our definition of coding experiments (Section 5.2.1), in which there are seven task categories: Design, Writing, Maintenance, Debugging, Testing, Inspection, and Comprehension. Table 22 shows the results when categorizing coding tasks in coding experiments.

Table 22 – Task Categorization

	Experiment		Task	
Task Category	Quantity	%	Quantity	%
Maintenance	33	41.25	39	31.45
Writing	19	23.75	34	27.42
Testing	9	11.25	19	15.3
Inspection	8	10.0	12	9.677
Comprehension	6	7.5	8	6.45
Debugging	3	3.75	7	5.65
Design	2	2.5	5	4.03
Total	80	100	124	100

The task categorization is not straightforward. A task in a category can also be classified in another category. For instance, Steffe e Thompson (2000) performed a coding experiment in which participants have to maintain the test suite in a project. Therefore, such task can be classified in both maintenance or testing categories. In such cases, an expert (the supervisor and co-supervisor) was consulted.

As we see in Table 22, the maintenance tasks are the most common tasks. We observed this phenomenon in both indexes (**Experiment Quantity** and **Task Quantity**). We attribute this phenomenon to the fact that maintenance tasks are the most costly activities in software development (PIGOSKI, 1996; SOMMERVILLE, 2004). Surprisingly, in Sjøberg et al. (2005), such tasks appeared as the third most common, after inspection and document comprehension.

Writing tasks are the second most common category (also in both indexes). Frequently, writing tasks are carried out together with other tasks. Usually, such tasks are designed to develop an initial version of a project in the experiment. After, subjects have to implement new features in this early version.

We identified nine experiments involving testing tasks. As said before, many experiments compared this technique with inspections. Several of them are also used to compare testing techniques, such as Pančur e Ciglarič (2011). Our results are in agreement with an SLR about testing (RAFI et al., 2012). However, it is contrasting with Sjøberg et al. (2005). One reason for this disagreement may lie in differences in definitions of testing.

Inspection tasks occurred in 8 experiments. The majority of them compared inspections and testing (such as Wilkerson, Jr e Mercer (2012)). In some other experiments, authors compared different inspection methods, such as Lucia et al. (2008)). In principle, our finding regarding inspections is not in concordance with Sjøberg et al. (2005). They observed inspections was the most common category. We propose an explanation for such discrepancy. During 1993-2002, software inspections were a new promising approach, requiring experiments to evaluate it. Nowadays, This technique is well evaluated, and it is considered reliable (GILB; GRAHAM; FINZI, 1993).

As we see in Table 22, comprehension task is the 5th most common coding task category. This rate is not in agreement with Ko, LaToza e Burnett (2015). We attribute this inconsistency to different definitions of experiments. Ko, LaToza e Burnett (2015) include case studies and other empirical studies as experiments. Moreover, as observed by Borges et al. (2015), case studies are the second most frequent empirical research in SE (BORGES et al., 2015), and many case studies investigate coding comprehension techniques. However, this result is a relevant issue for a meta-analysis.

In Table 22, debugging and design tasks had the lowest occurrences in both indexes. A low occurrence of design tasks was expected since such activities are out of our scope. We only found design tasks in experiments which such tasks did not take part of the evaluation, such as Phongpaibul e Boehm (2006). However, a low occurrence of debugging task is an alarming finding. Debugging activities are crucial in software development (LAWRANCE et al., 2013).

5.4.5.2 Task Duration and Ordering

We identified 48 coding experiments that reported an accurate report of task duration (60% of total). With this information, we classified each task in short-term and long-term tasks. Short-term tasks are those performed in one session that lasted from some minutes up to few hours. Long-term tasks are those in which participants were free to perform the experimental tasks at any time. However, they had a deadline. This deadline could be some days until up to months.

Figure 6 shows the distribution of the amount of time in the 25 experiments with short-term tasks. Eight experiments of total was measured in minutes. The remaining were measured in hours. On average, a task lasted 2.4 hours. The shorter task lasted 13 minutes, and the longer took 8 hours. We identified seven experiments with long-term tasks. As we can observe, there is a predominance of short-term tasks in coding experiments. It is desirable to avoid threats like tiring effect (WOHLIN et al., 2012).

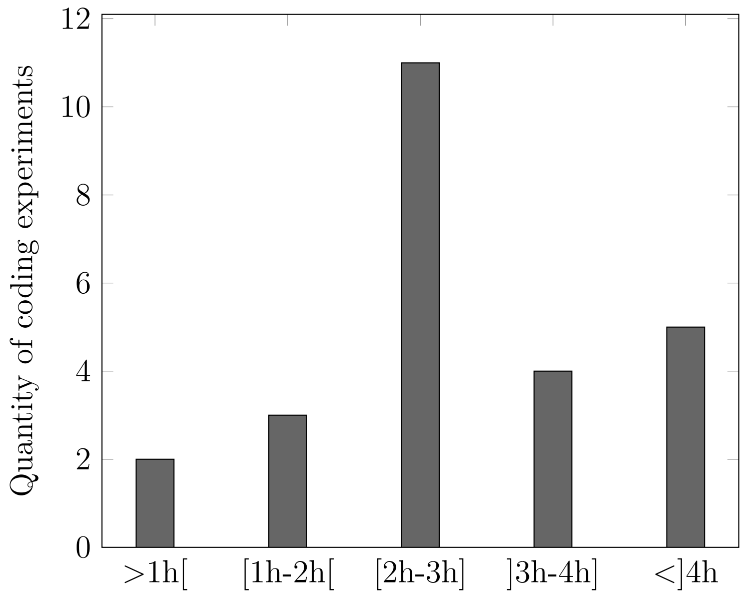


Figure 6 – Distribution of experiments with subject-level duration data to time intervals.

When designing an experiment, the number and order of tasks are crucial. In particular, the order is relevant since foreknowledge of next tasks may interfere in the time required to finish the current task. Such conditions can bias experiment results (WYLIE; ALLPORT, 2000). Regarding the number of tasks, a large section can also influence experiment results (i.e., tiring effect). Table 23 presents the strategies to order tasks in coding experiments.

Table 23 – Strategies to order tasks.

Ordering Strategy	#
Sequential	44
Unique	17
Alternated	10
Randomized	7
Exclusive	2
Total	80

In 17 coding experiments, participants have to perform only one task. In half of coding experiments, both treatment and control groups performed a set of tasks in a same order. On other ten experiments, the treatment and control groups performed a set of task in reverse order. It means if the control group performed the following task sequence T1, T2, and T3; then the treatment group performed the sequence T3, T2, and T1. In other seven experiments, participants executed the experimental tasks in a random order. Many guidelines recommend an alternated or random order since it can mitigate some threats, such as learning effect and expertise (MOSHEIOV, 2001). Finally, in two experiments, tasks performed by treatment and control group were different.

As said before, the quantity of tasks in an experiment affects experiment outcomes. Excluding those 17 experiments with only one task, the quantity of tasks in an experiment ranges from two to 19, however the majority of experiments has only two tasks. Finally, not only the order and number of tasks influence tiring effect. Task complexity also has to be taken into account. For instance, in the experiment reported by Wilkerson, Jr e Mercer (2012), participants had to perform a single task, but such task lasted three weeks. The authors observed an undesirable tiring effect in their results. However, more research is needed when designing coding experiment tasks.

5.4.6 Environments

In Section 5.4.4, we presented some confounding factors related to the environment where participants perform coding activities. However, a deep analysis of such factors is essential since artificially designed environments can bias experiment results. So that, results from experiments cannot be equivalent in industry. In this context, this section presents some findings about environments in coding experiments.

5.4.6.1 Location

There is a trade-off between realism and control regarding experimental task environments. On the one hand, a laboratory set up promotes an easier controlling and monitoring of experiment environment. On the contrary, in such a settings, the realism is decreased. Running an experiment in a usual office environment of subjects that are professionals allows a certain amount of realism, yet increases its fidelity due to breaks, phone calls, and other interruptions.

Regarding the 24 experiments with professionals, only 12 did not describe the context where the experimental tasks were performed. In other ten experiments, the authors informed that it was in a regular office. And, in two experiments, the authors explicitly stated that the experiment was in a laboratory.

In the 64 experiments with students, only four reports no explicit information about experimental settings. The other 60 experiments explicitly stated that they were performed in a laboratory or classroom.

All the experiments with students report the university/college's name (usually one of the author's institutions). In seven experiments that include professionals, they cited the involved companies' names. Nine experiments did not report organization's name. However, they report organization's business sector (transport, telephony, etc.). In fact, some companies have policies to assure that they remain anonymous in experiment reports. Finally, in five cases, the authors describe professionals as coming from “several” companies or organizations.

5.4.7 Observed Variables

A significant choice when designing an experiment is the definition of experimental task outcomes. Such outcomes are the observed variables. There are widespread approaches to specify them, such as GQM (SOLINGEN et al., 2002) or Six-Sigma (HARRY, 1998). In this section, we focus only on defining and measuring observed variable in a clear and accurate manner.

We found 318 observable variables in coding experiments. There is a broad range of variables, including task completion, time on task, failure detection, the number of tests, etc. Despite this broad range, we observed that three categories classify all of them: variables based on time, deliverables, and questionnaires. Table 24 presents the distribution of the observed variables in these three categories. In some cases, more than one category covers a single variable. Only variables based on questionnaires and deliverables are mutually exclusive. The next section details each category and their combinations.

Table 24 – Observable Variable Category.

Category	#	%
Only deliverables	125	39.31
Only Time	66	20.75
Only Questioner	34	10.69
Deliverables and Time	19	5.97
Questioner and Time	44	10
Others	30	9.43
Total	318	100

5.4.7.1 Time

Many observed variables in coding experiments measure the amount of time spent on a task. Some considerations are important when measuring them: (i) when the task starts and ends, (ii) what to measure, (iii) how to determine when the participant has finished, and (iv) data collection instruments.

First, experimenters have to determine the start and end point of each experimental task. For instance, the task starts immediately after delivering to subjects the task description and artifacts. In such experiments, the time to finish the task includes the time to read and comprehend the task description. It introduces some variability since each participant has a different reading speed. Besides, it also produces an uncertainty on whether they understand all the information, skimmed it or glanced at just the first few lines. Moreover, if the subjects have unlimited time to read the description task, they may start to plan the solution during the reading time. Such behavior can decrease artificially the time to finish the task. Usually, coding experiments measured the time after subject finishes to read the task description. However, some experiments documented, separately, the time to read and the time to perform an activity.

Time in coding experiments is not restricted to measure the time to finish a task. Some coding experiments measured the time of editing each file in the source code. In other experiments, authors made a distinction between the time dedicated to testing and implementing new features.

A challenge when measuring variables based on time is to define task ends. Many coding experiments solved this problem by providing a test suite. Therefore, a task only can be considered as accomplished only when all tests pass. However, the majority of the coding experiments did not adopt a particular approach to defining the end of their tasks. In such experiments, either the subjects or the experimenters decide when the task is completed.

Another important issue is to define means to measure time. Table 25 presents the most common means. Often, participants have to record the timestamp of start and end of each task. In some cases, they estimate how much time was spent on each task. In 17 cases, the experimenter measured the time by watching the participants. Finally, we observed that the long-term experiments usually gauge time via questionnaires, forms, etc. Only in eight experiments, we did not identify the mean used to measure the time.

Table 25 – Time Measurement Method.

Method	#
Form	42
Automatic tool	24
Experimenter	14
Tool and Participant	4
Experimenter and Participant	3
Not clearly defined	8
Total	95

Regarding automatic support, we found some initiatives to support coding experiments. Some researchers developed a complete tool to support their experiment (MURPHY; KERSTEN; FINDLATER, 2006; MÜLLER; HÖFER, 2007; ACCIOLY; BORBA; BONIFACIO, 2012; RIBEIRO et al., 2011; SALMAN; MISIRLI; JURISTO, 2015; MEDEIROS et al., 2017). Other researchers adapted general approaches, such as virtual machines, integrated development environments (IDEs), repositories, or screen recorders. Four coding experiments used a specific tool to support experiments in SE, the SESE (Simula Experiment Software Environment) (ARISHOLM et al., 2002a).

Finally, Wohlin et al. (2012) stated that it is desirable to use more than one means to collect data. Many data sources improve precision. However, only seven experiments used more than one ways to measure the time.

5.4.7.2 Resource

In coding experiments, many observed variables are measured by metrics calculated on deliverables (144 variables). Usually, participants produce some artifacts (for instance, source code). Based on these artifacts, metrics are calculated.

In 19 coding experiments, observed variables are a combination metrics over deliverables and the time spent to produce them. Frequently, researchers compare the quality of deliverables and time spent to produce them Wickelgren (1977). According to these authors, the time spent in a task impacts in its success. Therefore, participants may work faster and less carefully, reduce time, but also lower the quality of their work. Other participants may work more slowly and carefully, increasing time and success. In coding experiments, the majority of variables measured from deliverables are software metrics. Classifying found metrics is out of our scope. However, there is an extensive literature on this issue (CONTE; DUNSMORE; SHEN, 1986; BOEHM et al., 1981; KAN, 2002; PRESSMAN, 2005). Not all variables measured from deliverables are software metrics. In some cases, for instance, observed variables were a score by an expert of the source code produced by the participants.

5.4.7.3 Self-Reported Measurements

According to Wohlin et al. (2012), questionnaires are one the most common methods to collect data. In general, there are two types of questionnaires pen-and-paper or electronic (for example, e-mail or web pages) (WOHLIN et al., 2012). Table 26 reflects this fact. However, some experiments mentioned interviews and video conferences. There is an extensive literature about collecting data via questionnaires (FINK, 2003; JR, 2013).

Table 26 – Other Means to Collect Data.

Method	#
Pen-and-paper questionnaire	41
Electronic questionnaire	4
Report	1
Interview	1
Total	47

As presented in Table 26, variables gathered from questionnaires correspond to only 14.8% of observed variables. This low frequency may suggest an oversight of qualitative analysis in coding experiments. However, we observed that about only half of coding experiments perform qualitative analysis (44 experiments). To foster qualitative analysis in coding experiments, researchers can reuse practices from other sciences, such as usability (QUEIROZ; FERREIRA, 2009; BERNHAUPT; MIHALIC; OBRIST, 2008) or psychology (ORNE, 1962).

5.4.8 Replications

In our study, we identified seven papers that performed replications. Table 27 presents some characteristics of them, including their original experiments. This table is inspired on Sjøberg et al. (2005).

Table 27 – Code Experiment Replication.

	Paper	Exp	Stu.	Pro.	Con.	Rej.	Auth.	Rep. Type
	Phongpaibul e Boehm (2007)	0		X	-	-	-	
		1		X	X		Same	Different
	Bergersen e Sjøberg (2012)	0	X		-	-	-	
		1		X		X	Diff	Different
	Fucci e Turhan (2013)	0	X		-	-	-	
		1	X		X		Diff.	Close
	Vokáč et al. (2004)	0	X		-	-	-	
		1		X		X	Diff.	Different
	Arisholm e Sjøberg (2004)	0	X		-	-	-	
		1		X	X		Same	Different
	Acuña et al. (2015)	0	X		-	-	-	
		1		X	X		Same	Close
	Fucci et al. (2016)	0	X		-	-	-	
		1		X	X		Diff.	Close

As we see in 27 second and third columns, all replications performed by the same authors confirmed the results of their original experiments. However, we observed different results when other researchers authored them. Our results corroborate with other researches (SILVA et al., 2014; CARVER et al., 2014). They say that when the same people are involved in both original and replication, the results tend to be confirmed. However, we found a counterexample to this assumption (FUCCI; TURHAN, 2013).

In the last column, we see that among the seven replications only three were a close replication (LINDSAY; EHRENBERG, 1993), i.e., one attempts to retain, as much as possible, most of the known conditions of the original experiment. Other four replications were differentiated replications (SILVA et al., 2014).

5.4.9 Threats to Validity

A fundamental question concerns how valid are the results from coding experiments. Table 28 presents our findings regarding the report of the threats to the validity of coding experiments. As we see, 12 experiments (about 15.00% of all experiments) do not report clearly how the threats to validity were mitigated. In other 13 experiments (about 16.25%), the authors discuss this topic briefly.

Table 28 – Threats to Validity

	Generic	Specific Threat				Not Clear
		External	Internal	Construct	Conclusion	
#	13	56	54	36	18	12

In 68 coding experiments, we found a detailed analysis of their threats. Often, the authors address the external and internal threats to validity. We found only one experiment citing only the external validity and another citing only the internal validity. Moreover, about half of the 68 coding experiments that address the threats include a discussion about the construction validity. Also, only 26.47% of coding experiments discuss the conclusion validity.

Sjøberg et al. (2005) observed that 69% and 63% of the SE experiments discussed the threat to internal and external validity, respectively. Together with our findings in Table 28, the Sjøberg et al. (2005), Falcao et al. (2015)'s results is an alarming finding. Only an adequate mitigation of threats ensures valid results of an experiment for its population.

5.5 DISCUSSION

In this section, we discuss some implications of our SMS results regarding coding experiments in SE. We suggest strategies to aid future coding experiments, and we provide recommendations for future research.

Despiting the referenced guidelines, we observed fewer articles citing them. For instance, Wohlin et al. (2012) was the most cited guideline. However, it was adopted only by 5% of coding experiments. Moreover, about 30% of coding experiments did not cite any guideline. Some can argue that guidelines are common knowledge in SE community. Therefore, coding experiments are following them implicitly. However, Jedlitschka, Ciolkowski e Pfahl (2008) say that the quality of an experiment report decreases when it does not cite the adopted guidelines, tools, etc. On the other hand, Kitchenham et al. (2013) observed an improvement in the quality of some recent SE experiments. However, a sound statistical analysis is a lack in a threat in coding experiments (JØRGENSEN et al., 2015).

Another finding was an absence of guidelines for sub-domains in SE. As we presented in Section 5.4.4- 5.4.6, there are several characteristics specific only to coding experiments. Therefore, there is a lack of guidelines that consider such characteristics when designing and carrying out a coding experiment. Guidelines for sub-areas of acknowledgment is standard in other sciences like health (SCHULZ et al., 2010).

Regarding the foundation in experimentation themes, coding experiments adopted standard characteristics. Moreover, usually, coding experiments investigate a small number of factors, treatments, and hypothesis. Jørgensen et al. (2015) suggest avoiding studies with sophisticated design and many statistical tests. In particular, Jørgensen et al. (2015) incentive inclusion of few hypotheses and variables in the experimental study. A more straightforward design may reduce opportunities for researcher bias.

We did not find an agreement when concerning subject characteristics in coding experiments and generic SE experiments. About sample characteristics, we observed that coding experiments involved more graduate students and professional. We speculate that it is because many solutions evaluated in coding experiments regards problems from industry. Moreover, we observed certain standardization in coding experiments when describing their samples:

- Programming experience. Feigenspan et al. (2012) have found that self-estimation of programming experience on a scale of 1 to 10 correlates moderately with performance on Empirical Software Engineering Author's copy programming tasks (at least for undergraduate students). However, we believe that this estimation also may be applied to experienced participants.
- Industry experience. It is typically an indicator for multiple kinds of skills and knowledge, such as knowledge of version control systems, experience working with a team, and expertise with languages. In some scenarios, this information is relevant. However, the analysis cannot be restricted only to this information.
- Experience in a specific programming language. There are many programming languages and many paradigms of such languages. A previous acknowledgment of such languages and paradigm may affect the experiment results.

Other cited information describing samples in coding experiments are: language proficiency, location, age, and gender.

As we said before, one approach to enhance statistical analysis in coding experiments is performing uncomplicated experiments. Another approach is improving sample sizes. Therefore, it is crucial to recruit the right number of participants in coding experiments. However, recruiting is primarily a marketing problem of getting the attention of a set of people with specific characteristics. Therefore, new strategies have to be developed, such as distributed experiments (BUDGEN et al., 2013).

Our definition of coding experiment implies in a specific task classification. In fact, tasks in a coding experiment are quite flat. However, task design is at the heart of this tradeoff, as tasks represent a distillation of realistic and messy SE work into brief, accessible, and actionable activities. In coding experiments, about 80% of tasks regards writing new codes, maintaining them, or testing. Therefore, an interesting suggestion is to provide standards exploiting the environments where these activities are performed.

The physical or virtual location in which participants complete a task is another issue when designing experimental tasks. The task design affects the generalizability of experiment results. For example, a study performed in developer's actual workspace and toolset is likely to be more realistic than another when a developer is not familiar with the workspace and toolset. On the other hand, controlling developer's workspaces is a challenge. A well-known challenge is the influence of other participants. Considering an experiment involving multiple participants in the same room, participants might "follow the crowd" and finish early merely because other participants are leaving. If the study allows remote or asynchronous participation, these problems can multiply, because the experimenter might not be aware of interruptions or if the tool even worked correctly.

Another issue when designing coding task is the data and source code. Real projects and data from open source repositories ensure that the study results reflect real benefits. The downside to using realistic programs is that they can be disordered and complex. It can make the task more challenging for the study participants. They have to understand the project, requiring either more time in learning the system or introducing risks that the system will be too complicated for participants to complete a task. To work around this complexity, one can choose a real system, but focus participants attention (and tutorial materials) on a piece of the scheme that is less complex.

Overall more than 20% of the reviewed papers lacked any analysis of validity threats. It is an alarming finding. Empirical researchers should know that there is always several threats to the validity of any research study. Moreover, these threats should be examined both before and while designing the experiment as well as throughout other experimental phases.

Regarding replications, we realize that coding experiments and generic SE experiments do not diverge. Replications should be a general habit in SE community. Several researchers claim the importance of making replications (SILVA et al., 2014). Some initiatives have already begun to emerge as the scientific forums focused in replication [6].

Finally, this work did not stress to the maximum coding experiments. However, some interesting features were found. For example, we have identified a reliable standardization in data collection and task characteristics. These standardizations can be exploited to automatize some activities in coding experiments, like configuring environment and calibrate instruments to collect data. Some research in this sense has already been made (FREIRE

[6] International Workshop on Replication in Empirical Software Engineering Research.

et al., 2013).

5.5.1 Limitations

In the following, we present limitations and strategies to mitigate factors that may threaten our study's results.

5.5.1.1 Selection of Journals and Conferences

We surveyed six journals and conferences: Empirical Software Engineering Journal (ESEJ), Information and Software Technology (IST), Journal of Systems and Software (JSS), Transactions on Software Engineering (TSE), Evaluation and Assessment in Software Engineering (EASE), and Empirical Software Engineering and Measurement (ESEM). Some of them have a general-purpose in SE, other focus on the empirical studies in SE. Our selection of venues is similar to other studies (BORGES et al., 2015; KITCHENHAM et al., 2013). However, due to limited human resources, we could not cover the same venues as Sjøberg et al. (2005). This threat may had affected our results, however, considering a study with a broader set of venues (SJØBERG et al., 2005), our set of venues contained about 90% of all identified experiments.

5.5.1.2 Selection of Articles

We used a well-established definition of experiment and coding experiment to ensure an unbiased selection process (Section 5.3). Besides, we followed a strict multi-stage process to select and extract data. It involved several researchers, and the entire process is well-documented as suggested by Keele (2007).

Our search was organized as a manual search process of a specific set of journals and conference proceedings. It was consistent with the practices of other researchers looking at research trends as opposed to software technology evaluation. According to Kitchenham (2004), a manual search is recommended when researchers aim to map a specific acknowledgement area. Wohlin e Aurum (2015) obtained similar sets of primary studies when applying automatic and manual search.

Finally, the aim of this study is not to be exhaustive in coding experiment area. However, we believe that our research gives a good picture of the state-of-the-art.

5.5.1.3 Replicability

Replicability of systematic literature mapping studies is fundamental. It brings both trustability to produced results , as well as, opportunities to extensions. In our context, an important aspect is the reuse of previous studies (as presented in Section 5.3). We adopted this approach, since, considering all papers published in selected venues' set, they are about 50 thousands papers. Considering our available resources, it is not feasible to analysis all these papers. Moreover, the same approach of reusing sources of previous works was used in other researches (KITCHENHAM et al., 2013; KAMPENES et al., 2007). Finally, we also invite other researchers to replicate our study, in order to verify if they achieve same results.

5.5.1.4 Data Extraction

We presented the data extraction process in Section 5.3.4. The process was resource consuming and error prone. To mitigate this threat, we applied some techniques from Kitchenham et al. (2013). Besides, all data is available for auditing [7]. Moreover, some of our findings were analyzed via script in JavaScript (in particular the summing and counting activities), and they are also available [8]. A criticism to our data extraction approach is bias in our script. Due to a high quantity of produced data only part of the data produced by our scripts were verified manually, and no disagreement were found. Moreover, as said before, all data is available and we invite any researcher to replicate our analysis based on other approaches.

5.6 CHAPTER SUMMARY

This chapter provides a systematic analysis of the current status of the coding experiments in SE. In summary, coding experiments are any experiment (or quasi-experiment) in which individuals (or groups) have to conduct one or more coding activities. Coding activities are any activity related to the creation of code, maintenance, testing, inspection, comprehension, or debugging. We included articles published in renowned venues: two conferences and four journals.

Comparing our findings with other works in literature, we observed some agreements and disagreements. We believe our research reinforces conclusions from other works (SJØBERG et al., 2005; KO; LATOZA; BURNETT, 2015; HOST; WOHLIN; THELIN, 2005; JØRGENSEN et al., 2015) that the SE community needs mechanisms to provide better support to experiments in SE. This support has to consider both general SE experiment characteristics and those characteristics specific for a sub-area.

Our analysis did not stress the issues related to coding experiments. There is more to characterize of such experiment than we have described. For example, we have not addressed the characteristics of artifacts and instruments involved in coding experiments. Besides, other empirical methods can be investigated regarding coding activities. For example, case studies can produce a rich real-world data, such as how the industry handles coding practice.

[7] available at http://bit.ly/1LT7OPh
[8] available at http://bit.ly/1LT7OPh

Finally, this work contributes to thesis goals (Section 1.2) by providing a deep analysis of the current status of coding experiments. In particular, this works presents a description of how coding experiments are being carried out, specially regarding to subject description (Section 5.4.4). Besides, with regards to other characteristics we did not observed specific particularities in coding experiments. However, some fundamental findings were some standards in coding experiments, specially for tasks (Section 5.4.5) and involved variables (Section 5.4.7).

6 A METAMODEL TO GUIDE CODING EXPERIMENTS

The theory of our modern technic shows that nothing is as practical as theory.

—Julius Robert Oppenheimer

6.1 INTRODUCTION

In Chapter 4, we presented an analytical study of available MDE solutions to support experiments in SE. In summary, we have identified some common deficiencies in these solutions, especially in task description and instruments. On the other hand, Chapter 5 identified some existing practices in coding experiments, especially regarding data collection and tasks. So that, these standards can be exploited to automatize some activities in coding experiments, like configuring environment and calibrate instruments to collect data. However, models proposed by the MDE solutions did not specify adequately such standards.

An alternative to specifying experiments involving software development are languages tailored only to describe software development processes, such as BPMN (WHITE, 2004), BPEL (ALVES et al., 2007), and SPEM (SCHUPPENIES; STEINHAUER, 2002). In fact, these languages enable researchers to specify all experiment tasks involving coding activities (code, artifacts, tests, documents, and so forth.). However, they fail in specifying relevant information concerning experiment domain such as variables and measurements. In other words, they lack what MDE solutions support.

Despite the importance of each solution for SE experiments or software developments, to the best of our knowledge, there no modeling language that builds a bridge between software development and experiment processes. In this chapter, we address the research question of which concepts are common in these areas (SE experiment or software development processes) and how concepts can be used to guide coding experiment execution process. To this end, we investigate languages to specify SE experiments and languages to specify software development processes. Our contribution is a metamodel specifying only essential data to execute or replicate coding experiments according to their process specification. So that, our solution may allow (partial or fully) automation of some experiment procedures (such as environment configuration and data collection).

In a previous work, we proposed a meta-ontology to specify domain-specific ontologies for SE experiments (FERREIRA et al., 2015). As a proof of concept, we proposed an ontology specific for coding experiments. This ontology merged all common concepts in MDE approaches to specifying experiments and software development. However, we noticed that such specific ontology become easily verbose and redundant. Aiming at mitigating this limitation, we simplified this ontology. We call this new approach as Coding Experiment metamodel (CodEx metamodel). We decided to propose this new solution as a metamodel instead of ontology since our approach focuses on solution domain (CALERO; RUIZ; PIATTINI, 2006). We believe that models in compliance with our metamodels can be used by Integrated Development Environments (IDEs) to simplify the configuration and monitoring of coding activities. So that, leading to less intrusive coding experiment, more integrated experiments in user's daily work, as recommended by Wohlin (2013).

The rest of the chapter is composed as follows. Section 6.2 presents our research design. Sections 6.3-6.5 presents our results and how they were used in our metamodel proposition. Section 6.8 summarizes the chapter and offers suggestions for further research.

6.2 RESEARCH DESIGN

Considering the previous section, we conclude that there is a lack of standard specification for SE experiment and software development process. Aiming at fulfilling this gap, we analyzed coding experiments in literature and models for SE experiments and development process (Chapter 5). With this result, (1) we elicited meaning and develop knowledge into concepts used within both areas and (2) we identified patterns combining the identified concepts.

6.2.1 Methods

To address the first point, we gathered coding experiments and we analyzed each of them, together with the solutions presented in Chapter 4. Our goal in this phase was looking for how such solutions are being used. As a result, we generate a set of clusters which encapsulates all identified general concepts and their relationships. To address the second point, we adopted an approach based on Muehlen e Recker (2008). We created a spreadsheet, and we recorded each concept and its relationships with other concepts. We encoded their usage with 1 or 0. After, we applied a hierarchical clustering on the produced data to identify concepts that frequently or rarely occur together. Based on these results, we propose the Codex metamodel.

Depending on the research perspective, research results may have different interpretations. Our work targets the perspective of planning and carrying out a simple coding experiment. However, we believe that our results are relevant to other research perspectives, such as replications, family of experiments, and meta-studies.

6.3 MODELING APPROACH

After analyzing coding experiments in literature together with models for SE experiments and development process, we identified eight concept clusters:

1. *standards in empirical studies*: concepts shared by any empirical studies (DONOVAN; LAUDAN, 2012);

2. *goal*: any information describing results or possible outcomes;

3. *variables*: all variables involved (controlled or observed) in experiment process (WOHLIN et al., 2012);

4. *subject description*: experiment sample concepts;

5. *design of experiment* (DoE): concepts regarding the chosen DoE;

6. *tasks and activities*: any information describing experiment tasks;

7. *artifacts and instruments*: any objects used to carry out an experiment or any measurement instruments;

8. *validity evaluation*: any information about threats to experimental validity. More information about our clustering process is found in Ferreira et al. (2017a).

All previously cited concepts must be specified in any experiment (JEDLITSCHKA; CIOLKOWSKI; PFAHL, 2008). However, despite their importance in SE experiment, only some of them provide useful information for automation in coding experiment procedures. After analyzing each cluster content, we identified three clusters describing useful information for coding experiment procedure automation: *variables*, *artifacts and instruments*, and *tasks and activities.*

Initially, our metamodel incorporated all concepts in each previously mentioned cluster. They were organized in the same hierarchy level. The spreadsheet with concepts relationships pointed out few relationships between *artifacts and instruments* and *tasks and activities* clusters. On the other hand, these clusters are largely related with the *variables* cluster. This information was crucial when proposing each sub-package in our metamodel.

The following sections present the Codex metamodel concepts and architecture. Besides, we motivate each metamodeling decision using an illustrative running example. Finally, we take the opportunity to point out a mutual synergy between our solution and other solutions (Chapter 4).

6.3.1 Codex Metamodel Overview

Figure 7 gives an overview of Codex metamodel. The most basic entity in Codex metamodel is *Coding Experiment*. It compiles all main concepts identified by our analysis. As said before, we observed that all relationships between *artifacts and instruments* and *tasks and activities* clusters are via the *variables* cluster. Therefore, we created a package comprising *Artifacts and instruments* concepts, the **artifact** package (Section 6.4), and another package only for *tasks and activities* concepts, **task description** package (Section 6.5). All concept in *variables* clusters were defined inside the two aforementioned packages, and they build a bridge between these packages.

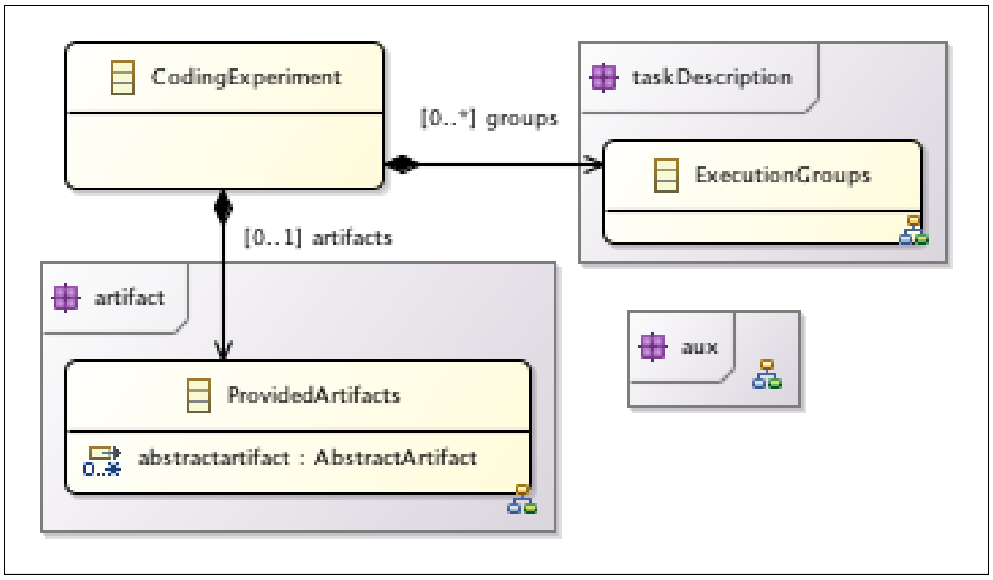


Figure 7 – Overview of Codex metamodel.

In Figure 7, the **aux** package does not define any particular experiment concept. It supports model concepts in other packages. This package contains the abstract class *Nameable*, which has only one attribute *name*. Therefore, any nameable concept in our metamodel does not have to define an attribute called *name*. It has to extend only *Nameable* class. Another abstract class is *Identifiable* (for classes with unique identification) and *Describable* (for classes with a description). Other metamodels adopt this metamodeling pattern, such as SPEM (SCHUPPENIES; STEINHAUER, 2002) and BPEL (ALVES et al., 2007).

6.4 ARTIFACT PACKAGE

When carrying out a coding experiment, a fundamental information is the artifact definition (such as source codes and tests). In a coding experiment, artifacts represents any article required to carry out a coding experiment activity. Figure 8 presents our artifact concept representation in Codex metamodel.

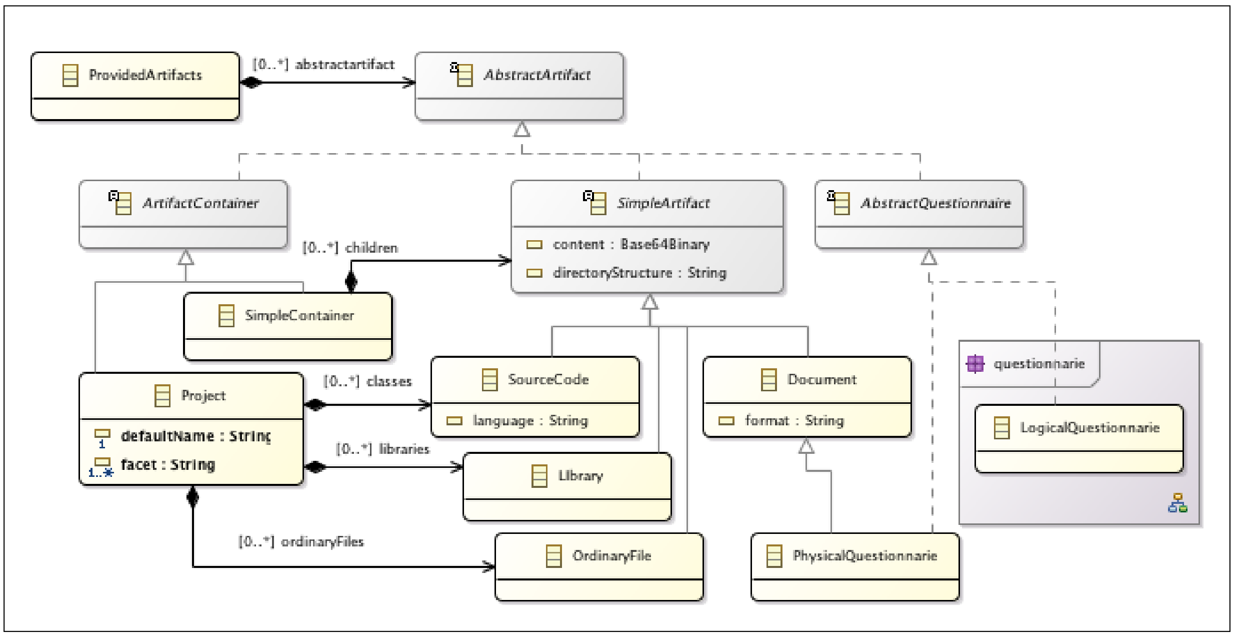


Figure 8 – Overview of artifact metamodel.

As shown in Figure 8, our metamodel classifies artifacts as *Artifact Container*, *Simple Artifact*, and *Abstract Questionnaire*. Next, we present the first two artifact types (*Artifact Container* and *Simple Artifact*). Due to its complexity to describe all elements in a questionnaire, we separated this concept in a specific package, **questionnaire** package. Section 6.4.1 details this package.

The abstract class *Artifact Container* represents a grouping of other artifacts (child artifacts). There are two means to assemble artifacts together, as a *Simple Container* or a *Project*. The *Simple Container* assembles child artifacts without any meta-information. For instance, in Stotts et al. (2003), participants had access to a zip file containing all archives required to carry out each coding task. A *Simple Container* is enough to model such cases. Another artifact arrangement is the abstract class *Project*. This class assembles child artifacts according to their coding project characteristics in an IDE. This class has two attributes: *defaultName*, it suggests an experiment project name; *facet*, it identifies each facet required to configure the subject IDE environment. Some *facet* examples are languages (Java, C, Python, and others), frameworks (Junit, Rails, Cucumber, and others), and infrastructures (Web development, mobile development, and others).

The abstract class *Simple Artifact* represents any atomic artifact. This class has two attributes: *content* which serializes the artifact content following base64 scheme (BORENSTEIN, 1993) and *directory structure* which describes any required structure (folder hierarchy) for an artifact. In Figure 8, we see our artifact classification. This classification depicts every possible artifact role in an experiment. Our artifact classification is according Cattaneo et al. (2000) and Silva e Oliveira (2011):

- *Source Code* comprises a collection of computer instructions, written using a human-readable programming language, usually as ordinary text. The attribute *language* identifies the source code's programming language (including tests);
- *Library* represents any artifact required to compile *Source Codes*. Usually, such artifact are a set of low-level routines used by a compiler to invoke some run-time environment behaviors;
- *Document* is an artifact to inform any crucial arrangement about the experiment. For instance, Santos, Mendonça e Silva (2013) made available a task description for each participant;
- *Other File* is any other artifact needed in any experiment task. For instance, in Stotts et al. (2003), the researchers made available to each participant a configuration file to start the experiment server.

Each source code class can seem like a compilation unit. In principle, a compilation unit is a section of text that can be submitted to the compiler, to create one or more modules of a program Venter (2007). We prefer to model this concept as a source code since there are many programming paradigms where this definition is not validly (for instance, logical paradigm). Moreover, we did not describe this concept in a fine-grained description, since this description is also changed in different programming paradigms. For instance, while OO programs have classes, methods, and attributes, logical programs have only clauses and facts.

6.4.1 Questionnaire Package

As anticipated in the previous section, a questionnaire is a particular artifact type. Questionnaires are one of the most common data collection means (WOHLIN et al., 2012). Questionnaires can both be provided in paper form or electronic (such as e-mail or on-line). We proposed three types of questionnaires:

- Physical Questionnaire. Those questionnaires has to be printed and filled manually by participants;
- Virtual Questionnaire. Those questionnaires are filled in web pages (such as, Google Forms and Survey Monkeys);
- Logical Questionnaire. In this type of questionnaire, each component in the questionnaire has to be described in details. Figure 9 summarizes our metamodeling approach for basic questionnaire elements. Our metamodel follows survey guidelines in SE (PUNTER et al., 2003; PFLEEGER; KITCHENHAM, 2001).

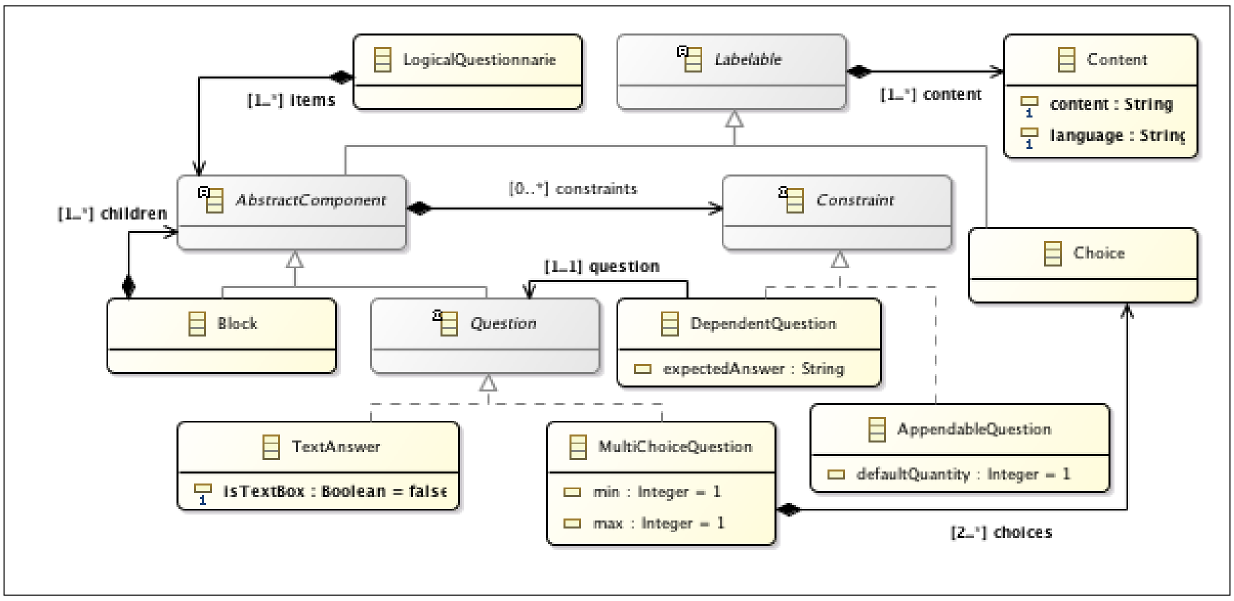


Figure 9 – Questionnaire metamodel overview.

Punter et al. (2003) define a questionnaire as a set of questions organized in a systematic way for the purpose of eliciting information from respondents. Moreover, questionnaires are classified as structured, semi-structured, or checklist (PUNTER et al., 2003). To model this variability, we proposed the abstract class *Abstract Component*. Any *Abstract Component* subclass shares some characteristic, such as a label. Unlike other solutions (such as ExpDSL (FREIRE et al., 2013) and eSEE (TRAVASSOS et al., 2004), a questionnaire component can have more than one label. Allowing many labels for a same component allows researchers to specify their questionnaire in different languages (KERSTEN; KERSTEN; RAKOWSKI, 2002). Next, we explain each subclass in *Abstract Component*.

The first *Abstract Component* type is *Block*. It comprises a set of other *Abstract Components*. Block organizes and modulates questionnaires. For instance, Santos, Mendonça e Silva (2013) used questionnaire blocks to group questions about demography and user experience.

The most fundamental *Abstract Component* type is *Question*. According to Azanza et al. (2010), there are more than ten question types. However, in our analysis, we observed two question types represent all questions in coding experiment questionnaires. *Question* types are:

- *Text Answer*. Such questions are designed to encourage a full, meaningful answer using subject's knowledge and feelings. The attribute *isTextBox* specifies if a response should be in few words (a text field) or a full paragraph (text box);

- *Multiple Choice Question*. In such question, participants have to choose their answer in a list of possible answers. The abstract class *Choice* specifies every possible answer.

Our metamodel also allows some question *Constraints*. The first question constraint is *Dependent Question*. This constraint in a question means this question requires specific answers from other questions. For instance, we can cite the Crime and Safety Survey Questionnaire (DEVOE et al., 2004). If the respondent is a female, she is given a Sexual Victimization Block apart from the Base Questionnaire. The second question constraint is *Appendable Questions*, which represents questions that can be answered many times. For instance, in Santos, Mendonça e Silva (2013), the participants had to identify classes in too much responsibilities, and for each class the participant had to answers a set of questions.

6.4.2 Synergy with Other Solutions

Some authors proposed metamodels only to specify Artifacts. Cattaneo et al. (2000) cover Web page artifacts, and Silva e Oliveira (2011) proposed an artifact metamodel to represent artifact parts and their relationships. These works are a convenient way to specify coding experiment artifacts. Such approaches inspired our concept of *Simple Artifact* (Section 6.4). Some SPMLs also propose an artifact specification. BMPN and SPEM classify artifacts according to their roles (input and output artifacts). The SPEM is the most accurate artifact representation with *FragmentDefinition* and *WorkProduct-Definition* (SCHUPPENIES; STEINHAUER, 2002). These definitions supported our concept of Artifact Content and Simple Artifact.

Concerning models to support SE experiments (Section 9.4.1), they usually specify artifacts as parameters or controlled variables. However, the ExpDSL and eSEE have entities to specify coding experiment artifacts. In ExpDSL, there is an entity called Artifact, which comprises a name, a description, and a type (input, output or input/output). This representation is similar to artifact specification in BMPN and SPEM. On the other hand, the eSEE proposed a complete sub-ontology to specify software artifacts (SANTOS; TRAVASSOS, 2007). It details a software artifact in a fine-grained description, differentiating interfaces, functions, and attributes. We have chosen not to follow this coding artifact description since this description is too bound with OO and imperative software paradigm.

To the best of our knowledge, there is no specific metamodel for questionnaires. However, some models to support SE experiments have some packages to specify questionnaires (Section 9.4.1). In ExperOntology, two classes (Questionnaire and Form) specify questionnaires. Questionnaire specifies demography data collection instruments and Form represents other questionnaires for nondemography data collection (ROOT; DRAPER, 1983). eSEE designs questionnaires in a sub-ontology for qualitative methods. Finally, ExpDSL proposes a view only for questionnaire specification. We tailored our metamodel to minimize variations among these models. Moreover, Codex metamodel is compliant with all these models. A simple M2M transformation can map our entities into other model entities.

6.5 CODING TASK PACKAGE

According to Software Engineering Body of Knowledge (SWEBOK) (BOURQUE; FAIRLEY et al., 2014), a coding or programming activity may vary from designing, writing, testing, debugging, and maintaining a system. Besides, activity duration also may differ in coding experiments. For instance, a coding inspection may be carried out on one occasion. On the other hand, writing an entire system can be executed during a much longer time span. In such cases, researchers cannot participate of all experiment activities and data collection.

Any coding task specification has to accommodate all possible contexts presented before. Aiming at such goal, we propose *Task* package into Codex metamodel (Figure 10). We defined one abstract class representing coding tasks and three sub-packages to describe them:

- *Task Order Package* defines how coding tasks are ordered and organized in coding experiment;
- *Task Constraint Package* specifies constraints on coding tasks, for instance, maximum amount of time spent performing a coding task;
- *Task Data Tracking Package* blueprints what data has been collected in each task.

The next sections detail each previously presented sub-package.

6.5.1 Task Order Package

This sub-package specifies task description, order, and organization. Figure 11 presents a metamodel overview. The first abstract class is *ExecutionGroup*. This class identifies which

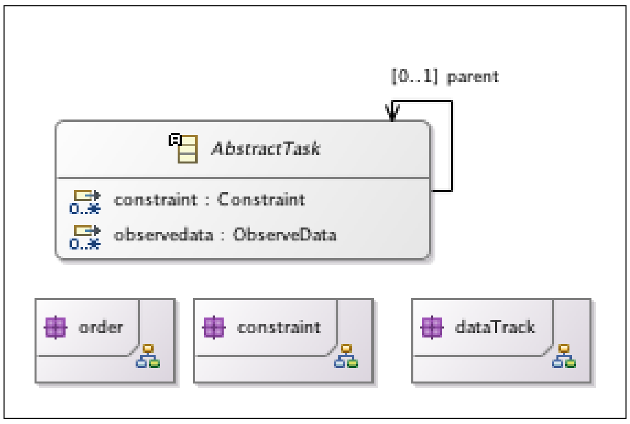


Figure 10 – Coding Task Overview.

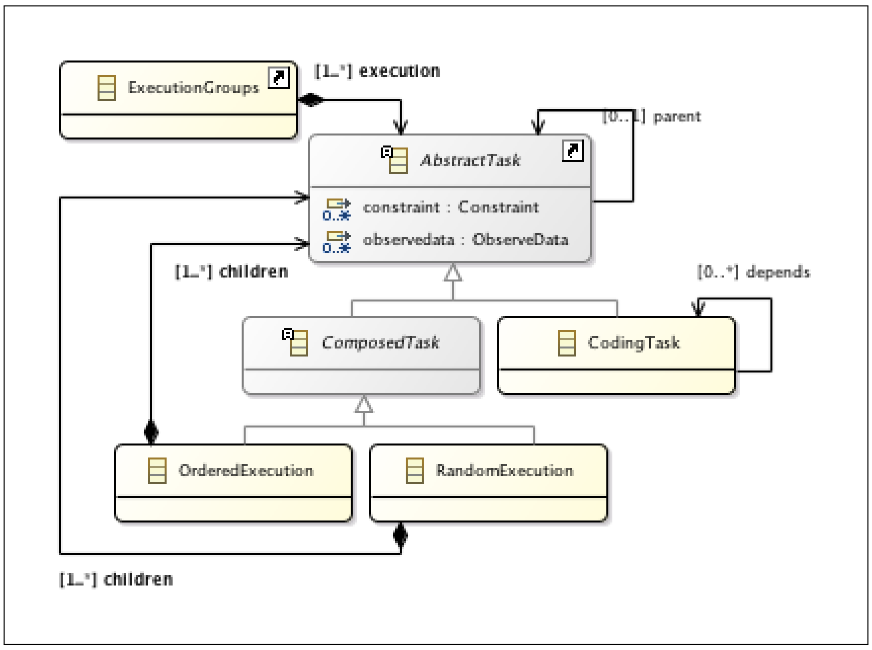


Figure 11 – Task Order Package Overview.

tasks has to be executed in each experiment trial. For instance, an experimental design AB has two *ExecutionGroup*, one indicating tasks *treatment A* and another indicating tasks with *treatment B*.

In our metamodel, task order specification is a BPEL and BPMN metamodel simplification (ALVES et al., 2007; WHITE, 2004). Therefore, there are two action types: *Coding Task* and *Composed Task*. *Coding Task* specifies atomic tasks in coding experiments. This class has an association called *depends*. This association identifies if a *Coding Task* requires artifacts or results from other tasks.

When participants have to perform more than one task, experimenters have to specify a *Composed Task*. This abstract class, in fact, is only an abstraction. We proposed two concrete classes for this abstraction:

- *Ordered Execution*: In this composition, tasks follow a predetermined execution sequence. This task composition is the most common composition type;
- *Random Execution*: This class represents executions without any predetermined order. *Random Task* executions are desirable to avoid some experiment threats (WOHLIN et al., 2012).

6.5.2 Task Data Tracking Package

In any experiment, data tracking is fundamental to control independent variables, as well as, to observe results in dependent variables (WOHLIN et al., 2012). Our metamodel allows tracking specification in most common coding experiment data sources. Figure 12 presents an overview of *Task Data Tracking* package.

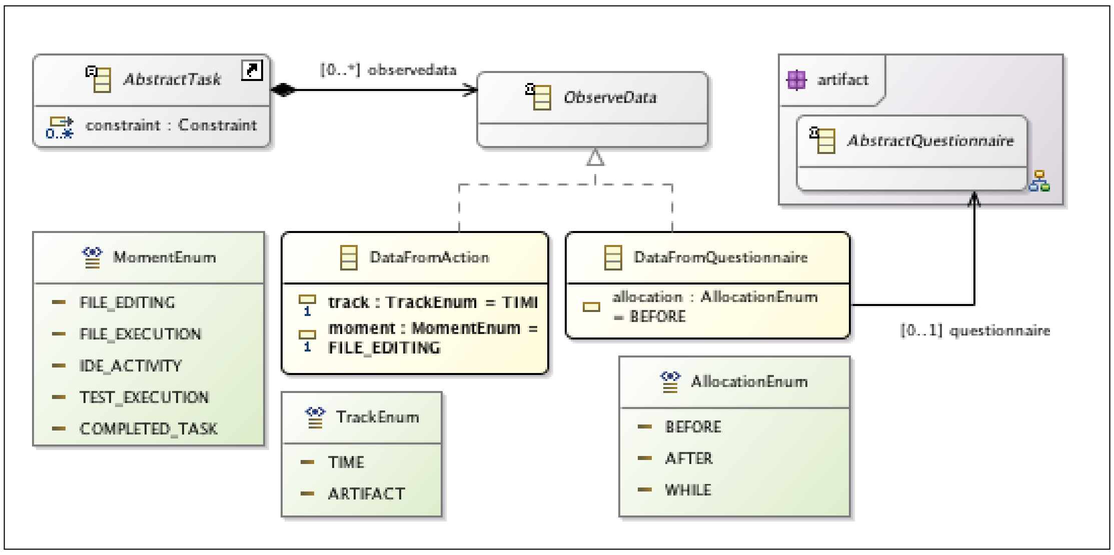


Figure 12 – Overview of Task Data Tracking Package.

Our metamodel specifies two data tracking means:

- *DataFromAction*: When data sources are participant actions. Usually, in coding experiments, two data types are tracked from participant actions: time stamp or involved artifacts. In our metamodel, *Track Enum* enumeration represents such data tracking types: *Time* and *Artifact*. *Tracking Actions* have another enumeration, *Moment Enum*. This enumeration specifies what action has to be tracked:
  - *Editing File*: Editing action in any (or a set of) Artifacts;
  - *Executing File*: When participants execute any (or a set of) Artifacts;
  - *Executing Test*: Similar to Executing File, when the participant executes test (or set of tests);
  - *IDE Activity*: Tracking any activity at coding development environment;
  - *Completed Task*: Data is tracked only at task end.
- *DataFromQuestionnaire*: When participants provide responses to questionnaires. The attribute allocation specifies when a questionnaire has to be administered. Moreover, each *DataFromQuestionnaire* is associated to at least one *Artifact Questionnaire*. We detail *Artifact Questionnaire* in coding experiments at Section 6.4.1.

6.5.3 Task Constraints

There are some examples in literature, which some independent variables in coding experiments may be specified as task constraints (SANTOS; MENDONÇA; SILVA, 2013; KO; LATOZA; BURNETT, 2015; VOKÁČ et al., 2004). In our metamodel, the abstract class Task Constraints specifies such constraints (Figure 13). In our analysis, we identified three common constraints in coding experiment tasks:

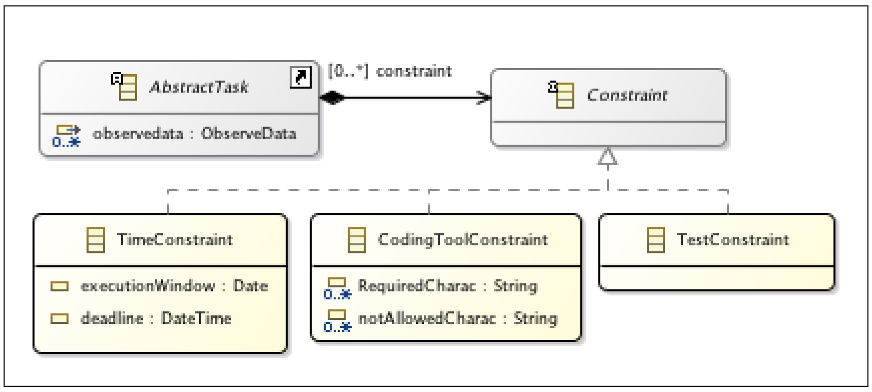


Figure 13 – Task Constraints Overview.

- *Time Constraint.* Time window constraint applied in any task in a coding experiment. It also specifies task deadlines;
- *IDE Constraint.* Constraint employed into coding experiment environment. In the current version, our metamodel only specifies required or forbidden Eclipse plug-ins. However, some metamodel extensions can include more constraints;
- *Test Constraint.* Defining success on task is not easy. Some examples in literature, researchers define a set of tests to ensure task success. Our metamodel implemented such scenarios with *Test Constraint.* This constraint is associated with a *Source Coding* (or a set of) identifying tests. Then, a participant only finishes a task when all test pass.

6.5.4 Interactions with Other Solutions

As said in before, BPMN and BPEL are our task definition and composition basements. In principle, *Abstract Task* is equivalent to *Activity* in BPMN. And, BPEL does not have any task representation. However, there is a standard extension, BPEL4People (KLOPPMANN et al., 2005) where Abstract Task is equivalent *WS-HumanTask*. Regarding the models to specify experiments, only ExpDSL and eSEE allow task definition and order. Moreover, similar to SPMLs, they are equivalent to *Abstract Task*. However, a surprising finding is a lack of random task order specification in both SPML and models to specify experiments.

Regarding data tracking, measurement is a central role in many standards and models such as ISO 15504 (ISO, 2004), including SPML and models to specify experiments. From the methodological perspective, software measurement is backed by a broad range of proposals, like the Goal Question Metric (GQM) method (BASILI; ROMBACH, 1988), the Practical Software & Systems Measurement (PSM) methodology (MCGARRY, 2002), and the ISO 15539 (EMAM; CARD, 2002) and IEEE 1061-1998 (COMMITTEE et al., 1998) standards. However, these are no clear link between these standards and experiment concepts.

Models to specify experiments (ESEML, ExpDSL, and ExpOntolofy) do not make a clear distinction between the observed variables, their measurements. Only eSEE specifies such information in Study Structure sub-ontology (SANTOS; TRAVASSOS, 2007). Furthermore, our measurement definition is a simplification of eSEE definition.

6.6 METAMODEL ASSESSMENT

The main goal of this section is to assess the our metamodel with respect to the modeling of coding experiments from the perspective of experimenters. To demonstrate the expressiveness of our metamodel, we developed a tool to model coding experiments according to our metamodel, *Codex Modelling Tool*. As said in introduction, this tool is part of a platform to support execution of coding experiments. Where, the Codex metamodel implements at core the concepts of coding experiments.

6.6.1 Assessment Method

We selected three experiments as examples of coding experiments in literature.

- Santos, Mendonça e Silva (2013). In this experiment, researchers designed a controlled experiment for investigating the concept of God Class, which is a class or methods with a high complexity. According to Fowler e Beck (1999), who introduces the concept of God Class is a class that tries to do much (i.e., it has many responsibilities and instance variables). Moreover, Ratiu et al. (2004) suggests a God Class detection technique considering those classes that use various data from the closer classes having the high complexity or low cohesion between methods. Based on this concepts, the objectives of In Santos, Mendonça e Silva (2013), this study is to find empirical support to evaluate the impact of personal conceptualization in god class detection;

- Accioly, Borba e Bonifacio (2012). This experiment were conducted with students simulating a test execution environment. While executing test suites, they collected time and reported change requests. This data was collected by an Eclipse plugin developed by authors.

- Vokáč et al. (2004). In this experiment, the researchers replicated the experiment performed by Prechelt et al. (2001), which investigated the question whether it is useful (with respect to maintenance) to design programs using design patterns, even if the actual design problem is simpler than that solved by the pattern. The replication sought to increase experimental realism by using a real programming environment instead of pen and paper, and by using paid professionals from multiple consultancy companies as subjects.

It is important to mention that our selection also considered the availability of documentation about experiment planning and conduction. The specifications of these experiments following our metamodel are available on-line[1].

We adopted one criterion for evaluating our metamodel, completeness. This criterion analyzes whether all concepts of coding experiments can be expressed in the Codex metamodel. For the completeness analysis, we investigated whether different aspects of the selected experiments can be properly specified using our metamodel. The following aspects from coding experiments were assessed during the specification process: *Tasks*, *Artifacts*, and *Measurements* (the clusters presented in Section 6.3).

6.7 RESULTS

In this section, we present and discuss the results of our study. Section 6.7.1 examines our findings regarding each cluster. Finally, Section 6.7.2 presents a discussion about our study and how it can be used by other researchers.

6.7.1 Analysis of each Cluster

The modeling of controlled experiments in our study revealed that the investigated metamodel satisfactorily addressed most of the evaluation criteria. On the other hand, it also exposed improvement opportunities for specific elements and aspects of the Codex metamodel.

This section presents results about the completeness criteria. We considered different experimental aspects that were modeled using Codex metamodel. We show and discuss how to model experiments in our metamodel. Also, we describe how the achieved results can be used to propose improvements for our metamodel.

6.7.1.1 Artifacts

Our metamodel allowed to define all the artifacts required to perform coding activities in each selected experiment. Figure 14 presents an overview of the model for artifacts in Santos, Mendonça e Silva (2013). Two types of artifacts were involved in this experiment, developing projects and questionnaires. Figure 14.1 presents a *Project* (Section 6.4). In this case, a Java *Project* which comprises each required source code (each java file). Originally, the researchers stored on-line all experiment artifacts. However, with a Codex model, it was no longer necessary as the Codex model stores this information together with all information required to carry out an experiment. The same approach was adopted by Vokáč et al. (2004) to specify their artifacts.

[1] https://github.com/netuh/DecodePlatformPlugin

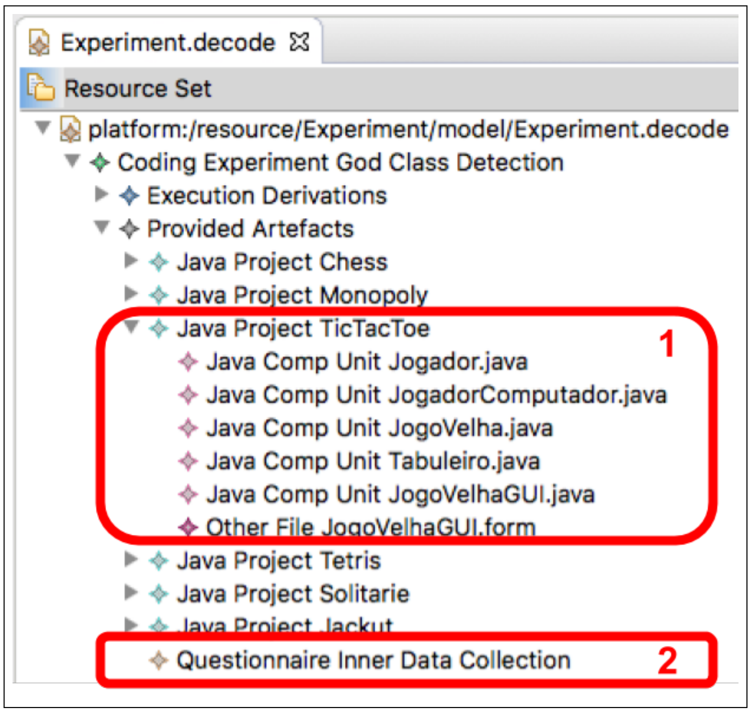

Experiment.decode
Resource Set
platform:/resource/Experiment/model/Experiment.decode
Coding Experiment God Class Detection
Execution Derivations
Provided Artefacts
Java Project Chess
Java Project Monopoly
Java Project TicTacToe
Java Comp Unit Jogador.java
Java Comp Unit JogadorComputador.java
Java Comp Unit JogoVelha.java
Java Comp Unit Tabuleiro.java
Java Comp Unit JogoVelhaGUI.java
Other File JogoVelhaGUI.form
Java Project Tetris
Java Project Solitarie
Java Project Jackut
Questionnaire Inner Data Collection

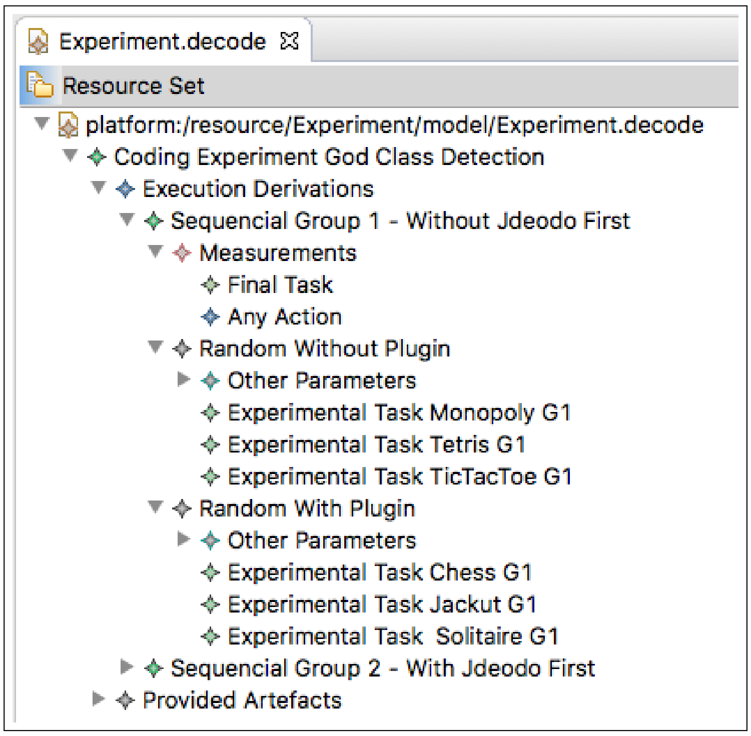
Figure 14 – Example of artifact specification.

Another artifact present in all selected experiments was questionnaires. Similar to previous mentioned Java artifacts, questionnaires are also included in Codex. Moreover, this model contains data about who and when to apply each questionnaire. Figure 14.2 presents how the questionnaires in Santos, Mendonça e Silva (2013) may be specified in our metamodel. In this case, a *Document* represents each questionnaire. However, researchers can specify each questionnaire element (according to Section 6.4.1).

6.7.1.2 Tasks

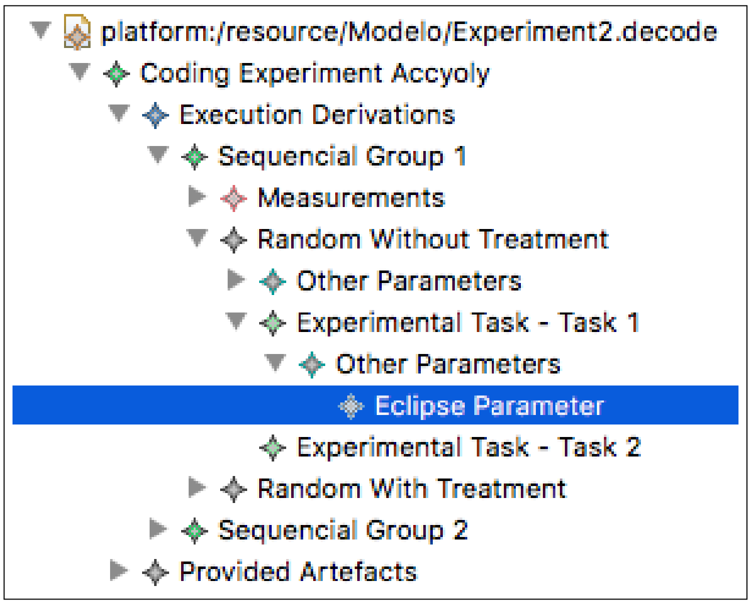
As we presented for artifacts, many relevant aspects of coding tasks were also defined in our metamodel. Figure 15 presents how the tasks in Santos, Mendonça e Silva (2013) are modeled in our metamodel. We decided to present the tasks in Santos, Mendonça e Silva (2013), since, in this experiment, part of the participants had to inspect three projects without any tool support and other three with a tool. These two set of tasks were executed in sequence. However, the three inspections with or without the tool were randomly executed. In Vokáč et al. (2004) and Accioly, Borba e Bonifacio (2012), all tasks were executed in sequence.

As can be seen in Figure 15, the derivation execution has two sequential tasks representing each trial in experiment design (*Sequential Group 1* and *Sequential Group 2*). For the sake of simplicity, Figure 15 only details the *Sequential Group 1*. However, all comments about *Sequential Group 1* are valid for *Sequential Group 2*. Moreover, each group was specified as an *OrderedExecution*; each *OrderedComposition* comprised two



```
Experiment.decode
Resource Set
▼ platform:/resource/Experiment/model/Experiment.decode
  ▼ Coding Experiment God Class Detection
    ▼ Execution Derivations
      ▼ Sequencial Group 1 - Without Jdeodo First
        ▼ Measurements
            Final Task
            Any Action
        ▼ Random Without Plugin
          ▶ Other Parameters
            Experimental Task Monopoly G1
            Experimental Task Tetris G1
            Experimental Task TicTacToe G1
        ▼ Random With Plugin
          ▶ Other Parameters
            Experimental Task Chess G1
            Experimental Task Jackut G1
            Experimental Task  Solitaire G1
      ▶ Sequencial Group 2 - With Jdeodo First
    ▶ Provided Artefacts
```

Figure 15 – Task specification in case study.



```
▼ platform:/resource/Modelo/Experiment2.decode
  ▼ Coding Experiment Accyoly
    ▼ Execution Derivations
      ▼ Sequencial Group 1
        ▶ Measurements
        ▼ Random Without Treatment
          ▶ Other Parameters
          ▼ Experimental Task - Task 1
            ▼ Other Parameters
                Eclipse Parameter
            Experimental Task - Task 2
        ▶ Random With Treatment
      ▶ Sequencial Group 2
    ▶ Provided Artefacts
```

Figure 16 – Task specification with test constraints.

RandomExecution. Finally, each *RandomExecution* included a set of *Coding Tasks*, connected with corresponding *Project* artifacts (Section 6.4).

Only Accioly, Borba e Bonifacio (2012) had constraints to task execution. Figure 16 presents how such constraints are modeled considering our metamodel. As we can see, researchers can attach a test suite to a task, *Test Constraint*. It means that the task may be considered as finished only when such tests pass.

6.7.1.3 Measurements

In Codex metamodel, we can define measurements for each task. During the data collection procedure definition (process element), the researcher can choose to collect data related to dependent variables using our metamodel in different ways, such as: (i) collecting artifacts created or changed during the experiment that should be analyzed by the researcher; (ii) requiring participants to inform the specific data (e.g. number of defects found) during their tasks execution in the experiment; or (iii) including a field in a questionnaire (through cross-reference) to be answered by the participants during the execution of the experiment.

Considering our example Santos, Mendonça e Silva (2013), Figure 15 presents how a researcher can specify measurements in tasks. In this experiment, two kinds of data have to be collected, (i) time spent to finish each task and (ii) quantity of actions executed by each participant. The first measurement was specified by a *DataFromAction*, with *Time* at track field and with *IDE_Activity* at moment field (details in Section 6.5.2). Similarly, for the second measurement, the corresponding *DataFromAction* has *Time* at track field, and *Final Task* at moment field.

The other two experiments (Accioly, Borba e Bonifacio (2012) and Vokáč et al. (2004)) collect almost the same kind of variables. However, they also collect the artifacts produced during the experiment execution. Therefore, their models have an extra element: *DataFromAction* which is attached to *Artifacts* and *Completed Task*. As so, once the tasks are completed, all artifacts (produced and changed) during the task execution are part of the results.

6.7.2 Discussions and Lessons Learned

In this section, we present and discuss lessons learned related to the results of our study.

Appropriateness Evaluation. In our study, we have tried to address such criterion by specifying three different experiments, among which we modeled one replication Baldassarre et al. (2014). Given this variability of experiments, we have modeled experiments: with diverse experimental designs and executed by different research groups. This variety of the chosen experiments supports us to conclude that the Codex metamodel provided an appropriated characterization for the set of modeled experiments. However, we recognize that it is crucial to model an extensive amount of additional experiments to enlarge the variability of experiments that can be expressed by our metamodel.

Experiment Replication. In spite there are various guidelines for reporting experiments, the fact that they have not been formalized according them limits their replicability Baldassarre et al. (2014), Juristo e Vegas (2009). In collaborative experimental research, results from previous studies are needed to transfer knowledge between the involved researchers. In this context, our metamodel provides a mean to formalize the laboratory package for coding experiments and their correspondent replicas. It is one of the tangential contributions of our approach. We believe that it could facilitate information communication and exchange among experimenters, contributing to fill the gap related to providing a complete experiment definition. We modeled one experiment replication, and we observed that when modeling a replica, a researcher can reuse almost all original experiment specification. The changes are localized and related to the following actions: to add or change artifacts, tasks, and/or measurements. We intend to systematically evaluate and explore the Codex metamodel benefits regarding replications in our future work.

Execution Environment. The experiment knowledge formalized with our metamodel is used in our approach Ferreira (2014) as input to an experimentation support environment. The environment allows executing an experiment as a workflow that guides the tasks of each participant. The environment supports some experiment functionalities:

- Experiment Documentation – As said before, other approaches are most appropriated to document an experiment in SE (FERREIRA et al., 2017b). However, our metamodel allows a precise description of many relevant characteristics in an experiment plan. This description should provide a formal way to store the plan of several kinds of experiments in software engineering to increase the documentation precision. For our execution environment, the experiment definition in Codex metamodel itself represents this functionality.

- Participant Guiding – We developed a tool to aid participants in their experiment execution. The environment guides participants through each task that have to be performed by them. The participants have to finish one task after another (sequentially). The experiment's Codex model specifies the task order.

- Data Collection – Each participant produces data that are relevant for the experiment. This data is automatically collected by the environment, such as changed files, or project execution, test passes and failed, and so on. Besides, the environment may provide a means for participants to answer questionnaires (according to their specification). Moreover, the environment is responsible for automatically collecting the spent time for executing each experimental task.

- Gathering of feedback – It is fundamental to gather feedback from participants during experiment execution. A questionnaire is a known instrument to collect feedback. The participants are asked to answer online feedback questionnaires according to the Codex questionnaire definition.

Limitations. Regarding the models to specify extensive experiments in SE. Our metamodel is able to specify all characteristics identified in cluster. However, we should conduct interviews with researchers from other companies and universities to try to understand their views about these models, as well as in other fields, and carry out experiments in practice. For instance, an information relevant to be specified is that only code without errors or faults cannot be accepted when finishing a task.

6.8 CHAPTER SUMMARY

Regarding the research questions under investigation, which concepts are in common with these two areas (SE experiment or software development) and how they are related to each other. Our study has initially identified a necessity for a unified metamodel for coding experiments since other alternatives do not have enough expressiveness. After an individual analysis of these alternatives together with real coding experiments, we have proposed a new metamodel, Codex metamodel. This metamodel provides a precise panorama focusing only on coding experiment executability. This metamodel identifies core elements that enable an automatic support to coding experiment execution. A customized model following our metamodel emphasizes variables, tasks, and artifacts involved when carrying out a coding experiment. It may foster coding experiments since experimentation is a cost-intensive empirical method. Therefore, by facilitating the experiment execution, the researchers may focus more efforts in improving samples, allowing more reliable conclusions (JØRGENSEN et al., 2016). In next chapters, we present how a tool support can use our metamodel to support coding experiments.

7 CODEX PLATFORM: A PLATFORM TO SUPPORT CODING EXPERIMENTS

He knew that all the hazards and perils were now drawing together to a point: the next day would be a day of doom, the day of final effort or disaster, the last gasp.

—J.R.R. Tolkien

7.1 INTRODUCTION

Experiments help researchers to make new and important discoveries in science. In SE, an experiment may investigate factors which influence two aspects of the software maintenance process; namely, understanding an existing program and accurately implementing modifications to it. A tested hypothesis becomes part of scientific knowledge if it is sufficiently well described and constructed, and if there is convincing evidence. However, experiments are often unfeasible in SE due to the high cost of projects (ROST; GLASS, 2011).

As we presented in Chapter 4, over the last years, the SE community has proposed different approaches to aid the growth of experiments. Such approaches include guidelines, tools, infrastructure, and many others (BORGES et al., 2015; WOHLIN et al., 2012; JURISTO; MORENO, 2013; BASILI, 1993). We focused on models and specification. However, these models are only part of MDE platforms to support SE experiments.

The term "platform" is unclear in SE literature. In this work, we adopt Wolf (2003), "a platform is a pre-designed architecture that designers can use to build systems for a given range of applications". In such platforms, models specify key design decisions about the problem that primarily govern its structure, behavior, and quality. In our context, such models are an experiment blueprint so that it can serve as the basis for experimentation (WOHLIN et al., 2012).

As presented in Chapter 1, the focus of this thesis is on coding experiments. In some coding experiments, the authors had to develop their own platform to run their trials. In some coding experiments, the authors had to develop their own platform to run their trials. For instance, Müller e Höfer (2007) compared test-driven development when performed by experts and novices. The authors developed a platform that comprised two Eclipse plug-ins. One plug-in captured timestamps and stores backup copies whenever a file is modified. Another plug-in logged the time when each test was executed, the number of executed tests, and whether it passed or failed. However, not all coding experiments had to develop their platforms. For instance, Wang e Arisholm (2009) performed an experiment where subjects had to perform some maintenance tasks. The authors used the Simula

Experiment Support Environment (SESE) tool to manage experimental artifacts in the subject environment. Each participant used the SESE to download experiment's codes and task descriptions. Moreover, the SESE was able to collect data some data relevant to the coding experiment.

As we can see, coding experiments employ a range of platforms. However, each platform plays a different role in different contexts. Some platforms usually are too specific and restricted to a experiment in a particular contexts. Alternatively, some platforms are too general that only generic information is collected and analyzed.

Considering these limitations, we propose Codex Platform, an extensible platform designed to support only coding experiments. It brings together benefits from both general and specific platforms. We offer an extensible and integrated representation for coding experiments using Codex Metamodel (Chapter 6). This representation focuses on essential information to configure and monitor coding activities. Moreover, Codex platform provides an environment for controlling and observing coding activities. Using our platform, researchers can design and execute quantitative and exploratory coding experiments with less efforts to configure and observe each trial.

7.2 CODEX PLATFORM OVERVIEW

In this section we present Codex Platform. We designed and implemented this platform, an extensible platform targeted to aid in carrying out coding experiments. Figure 17 presents an overview of Codex platform's architecture. This platform hinges around a two-tier architecture to separate the concerns of design and experiment execution.

The numbers in Figure 17 represent the steps necessary for designing and conducting a coding experiment in Codex platform. These steps are:

- **Step 1: Experiment Modeling**. The primary objective of this step is to allow researchers endeavor to model conceptual aspects in coding experiments. Experiment guidelines and documentation can support activities in this step. The final result is a model, the Codex Experiment Model, specifying all operational characteristics to execute or replicate each coding activity;

- **Step 2: Loading Experiment**. In this step, researchers have to recruit subjects that should take part in the experiment. Defining and recruiting subjects are still a researcher duty. After recruitment, each participant loads the Codex Experiment Model provided by the researchers in our Codex Eclipse plugin. This plugin is responsible for configuring participant's IDE automatically. Moreover, this plugin also profiles the experiment coding activities performed by each participant. The Codex Experiment Model provides parameters for tuning the configuration and profiling;

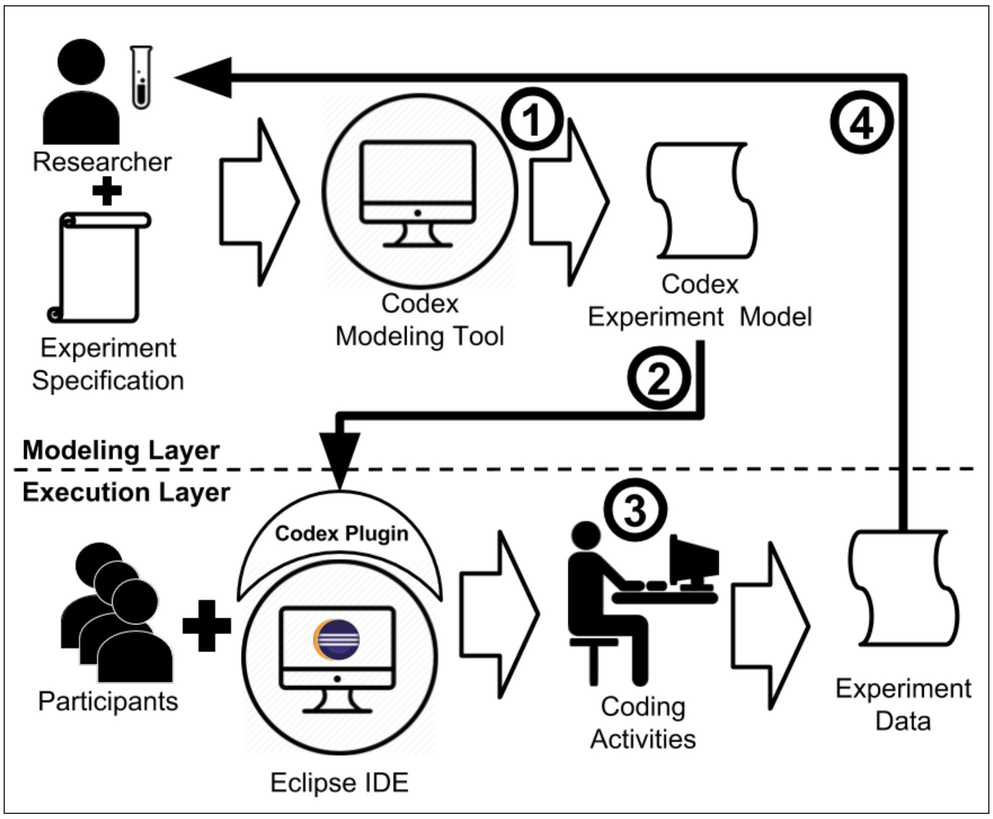


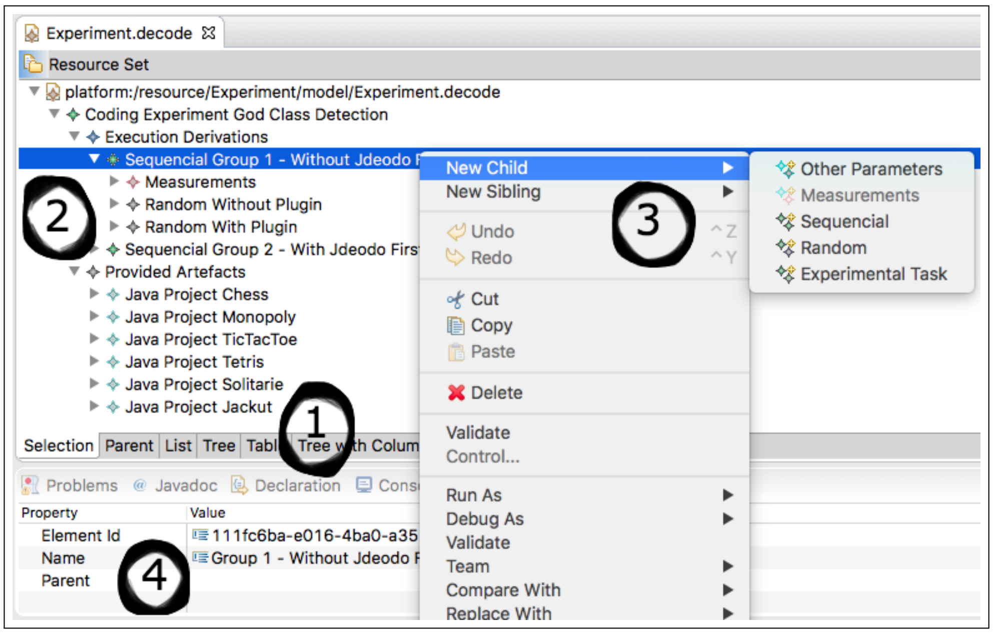
Figure 17 – Codex Platform Overview.

- **Step 3: Executing Experiment Tasks**. After configuring participant's IDE, the participants can execute each coding task. The task's characteristics and order follow the loaded Codex Experiment Model (designed in Step 1);
- **Step 4: Experiment Result**. After all coding activities, the Codex Eclipse plugin serializes the collected data. The serialized information has to be sent to researchers start the analysis.

Regarding our layered architecture (Figure 17), only Step 1 is at modeling layer. The remaining 2-4 steps are at Execution Layer. The next sections describe how our platform details about each tool involved in our platform.

7.3 CODEX MODELING TOOL

Our platform comprises multiple plug-ins for Eclipse Platform. The first plug-in is the Codex Modeling Tool. To develop and design this tool, we followed the Eclipse Modeling Framework (EMF) (STEINBERG et al., 2008). Its basement is Ecore meta-modeling language which is used to specify abstract syntaxes for arbitrary modeling languages. Thus, our modeling solution space is not bound to any particular language, and existing EMF-based modeling tools can easily be integrated. A Codex Model can be generated (partially



Property	Value
Element Id	111fc6ba-e016-4ba0-a35
Name	Group 1 - Without Jdeodo F
Parent	

Figure 18 – Codex Modeling Tool Screenshot.

or entirely) from other models to specify SE experiments such as eSEE (TRAVASSOS et al., 2004) and ExpDSL (FREIRE et al., 2013).

Figure 18 shows the Codex Modeling Tool panel. Each number in Figure 18 corresponds to a component:

1. **Toolbar**. It provides options to visualize Codex Models: Parent List, Tree, Table, and Tree with Columns. The default visualization is in Tree. Details about each option is in EMF guidelines (STEINBERG et al., 2008);

2. **Workspace**. It presents a graphical representation of a Codex Model. Each line represents an entity in our meta-model. In this view, researchers can expand each entity to see its child entities. This model is in compliance Codex Metamodel (Chapter 6);

3. **Drop-down Menu**. This menu pop-up is activated by clicking at right button mouse. The pop-up presents all available child entities according to with our Codex metamodel. Besides creating child entities, this menu has an option to validate the model under construction. In particular, it verifies if the model is syntactically validity regarding our metamodel. For instance, it verifies if an entity with an association "1..*" has at least one child entity;

4. **Property View**. While the previous item creates new child entities, **Property View** allows researchers to modify contents in an entity's attribute. Some attributes cannot be modified. For these attributes, this view presents only its content.

7.4 CODEX EXECUTION PLUGIN

Once the researchers had completed the recruitment, each participant can proceed in the experiment and start experimental tasks. Figure 19 presents the Codex Execution Plugin interface to carry out a coding experiment task. In particular, the user interface (UI) comprises four buttons. Each button implements one of the following activities:

1. Loading and Selecting Experiment. This button brings a new window presenting all available coding experiments. In this window, a participant can start a previously loaded experiment or load a new experiment. Section 7.4.1 details experiment loading process;

2. Configure Current Task. This button is only enabled after loading or selecting a codex experiment by the previous UI. When enabled, this button is a trigger to a set of actions to adjust the participant's Eclipse Workspace according to task specification. For instance, it creates an Eclipse Java project for each project modeled in the Codex model;

3. Start Current Task. This button is not enabled until a codex experiment is configured (the previous UI action). When a participant presses this button, our plugin starts to observe participant actions. What and how to observe participant action are performed according to the Codex model.

4. Finish Current Task. This button has two duties. Firstly, it stops all observers that were profiling participant activities. After, each monitor stores the collected information. If there are remaining tasks to perform, our tool identifies the next task. After identifying next task, our platform enables the button Configure Current Task again. This process repeated until there is no more task remaining in Codex model. When there are no more tasks to perform, our plugin presents a file selector window. The chosen file has to store all data collected from participant trial. Section 7.4.2 details data collection and storage process. Finally, the researcher can gather the file with the collected data from each participant separately.

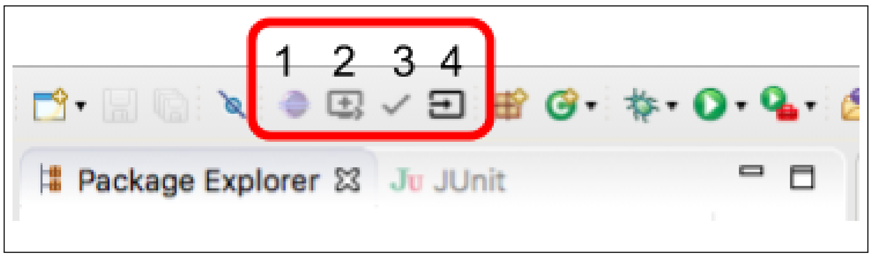


Figure 19 – Codex Execution Plugin Screenshot.

7.4.1 Loading and Selecting Experiment

This section presents the feature Loading and Selecting Experiment. The fourth button in Figure 20 triggers this feature. After pressing this button, the window in Figure 20 appears. This window comprises three main graphic elements:

1. Load New Codex Experiment Button. This button is at the top-left in Loading and Selecting Experiment window. When a user clicks on this button, a file selector window is displayed. In this file selection, a user has to select the Codex Model corresponding to experiment from the local file system;

2. Loaded Experiment Table. Each line in the table (Figure 20) represents a previously loaded experiment. The information presented on each line is the experiment's name and its status icon. There are three available status icons: Experiment Loaded, Experiment Started, and Experiment Finished;

3. Select and Cancel Buttons. These buttons are at the bottom-right in Loading and Selecting Experiment window. The Select Experiment button is only enabled after a user selects one experiment at Loaded Experiment Table. After selecting the corresponding Codex model, each participant has to choose an execution track. An execution track corresponds to a trial at the design of experiment (DoE) (WOHLIN et al., 2012). For instance, an experiment 2x2 usually has two tracks (control and treatment tracks).

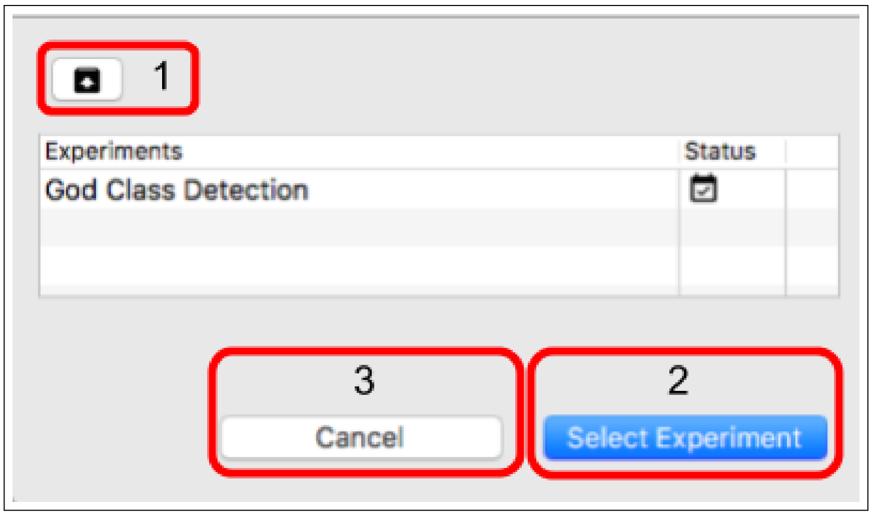


Figure 20 – Codex Execution Plugin Screenshot.

7.4.2 Collecting Data

Our platform monitors the experiments in real time. The collectible data depends on experiment specification in its Codex model. It may range from the total time spent to complete a task to artifacts created during the experiment (Section 6.4). To support the data collection, we used the Usage Data Collector (UDC) (FOUNDATION, 2018). It is a framework for collecting usage data information. It is a lightweight framework that installs listeners on various aspects of the Eclipse workbench, and—from those listeners—gathers information about the kinds of things that the user is doing (i.e., activating views, editors, and others). However, the current version of UDC is deprecated, so we had to update almost all UDC source code.

Regarding data collection, the UDC is a core element in our platform. For instance, considering Huang e Holcombe (2009)'s experiment, one of the variables observed in this experiment was quantity of test executions. This variable can be modeled by the entity *DataFromAction* in our metamodel (Chapter 6). We have to set *track* field to TIME and *momment* field to TEST_EXECUTION. When this entity is loaded in our Eclipse plugin, the UDC is set to listen all events about file execution and it has to record only those when execution file is a Java test (JUnit sub-class). All this configuration is automatic, the participant only have to load the experiment model before start a trial execution.

After a trial, the data collected from each participant is saved in a file. This file is in JSON File Format (SMITH, 2015). We decided to use this format, because statistical analysis tools handle such files, such as SPSS (GREEN; SALKIND, 2010) and R (DALGAARD, 2008).

7.4.3 Limitations

One of the criticisms of our design is creating two distinct buttons to configure and start tasks. A simple button could implement both features. Then, pressing this button, it triggered the operations for configuring and monitoring. We decided to divide these two actions separately in two buttons, because, in some scenarios, the configuring operations cannot be entirely automatic, for instance, Müller (2007). So that, we allow that after the automatic configurations some procedures can be carried out before starting each task. However, as we discuss in Chapter 8, usability is an issue in our platform. In particular, usability is a vital component of product quality and it becomes increasingly important once the initial excitement of a new technology dies down and customers look for effective use rather than technological novelty.

Some also may criticize our adopted architecture. Our architecture did not exploit a client-server Web architecture nor cloud computing. In such architectures, we need a set of servers to host part of the platform. It brings an overhead to configure the experiment platform. On the other hand, the goal of our platform is to reduce as much as possible the effort to carry out a coding experiment. Moreover, in some coding experiments, access to the internet is forbidden. In such cases, an offline platform is preferable. However, we plan as future work make some tests with such architectures.

In the context of Cloud Computing, the Eclipse Foundation released in 2016 the Eclipse Che (Eclipse Foundation, 2016), an open-source Java-based developer workspace server and cloud IDE which provides a remote development platform for multi-user purpose. It also contains an SDK which can be used to create plug-ins for languages, frameworks or tools. Unfortunately, the capability to create plug-ins in Eclipse Che was not available during the development of our tools. However, we plan to redevelop our Codex Modeling and Execution Plugins for the Eclipse Che.

7.5 DISCUSSION

As presented in previous sections, Decode Platform supports activities in planning and executing a coding experiments. However, benefits from our platform are not confined only in such activities. By adopting our platform, researcher mitigate some threats in their coding experiments. This section presents some common threats to valid of coding experiments and how our platform aims to mitigate such threats. Experimental validity refers to the manner in which variables that influence both the results of the research and the generalizability to the population at large.

In the following, we present some threats to validity and how our platform mitigate them:

- Hawthorne effect. This threat is also called as observer effect, it is a type of reactivity in which individuals modify an aspect of their behavior in response to their awareness of being observed (MCCARNEY et al., 2007). Our platform aids to mitigate such threat by automatically configuring and observing variables. So that, participants may feel that their activities are part of their daily activities, since they are using their common development context (IDE, projects, etc.). However, participants are affected by this effect, since they have to install and load experiment package.

- Fatigue effect. In psychometric studies of cognitive performance that last several hours, there are two major potential influences on performance (SUESS; SCHMIEDEK, 2000). Fatigue and loss of motivation may lead to a drop in performance while effects of practice on similar tasks can possibly help to improve results. Besides the effects on mean scores there is also the possibility of a change in the reliability or in the relations between single tasks and the constructs intended to be measured, thus an effect on validity. Our platform provides mechanisms to mitigate this effect, since participants can pause the experiment execution. However, this feature is only provided in the moment after finishing a task and before starting the next.

- Reliability of measures. The validity of an experiment is highly dependent on the reliability of the measures (WOHLIN et al., 2012). A manner to mitigate this threat is by data triangulation. Data triangulation is any action toward validates data and research by cross verifying the same information. This triangulation of data strengthens a research paper because data has increased credibility and validity. And our platform support this activity since, it collect data automatically via multiple sources. Besides, researchers are motivated to use other data source instead of relying only on the platform.

- Forecast Task Quantity. If participants know how many tasks they have to perform in an experiment, they may perform the current task faster or slower (WOHLIN et al., 2012). If a participant know that the current task is the last task, she/he may perform faster than normal pace to finish the experiment trial. The same may occurs if participants know that there are many other tasks. Our platform mitigate this threat by not showing the task quantity to participants, they only know the current task.

- Instrumentation. This is the effect caused by the artifacts used for experiment execution, such as data collection forms, document to be inspected in an inspection experiment, etc (WOHLIN et al., 2012). If these are badly designed, the experiment is affected negatively. This threat is related to the previous threat. So that, the same support provided for the previous threat also support the mitigation of this threat.

- Compensatory rivalry and Resentful demoralization. A subject receiving less desirable treatments may, as the natural underdog, be motivated to reduce or reverse the expected outcome of the experiment (WOHLIN et al., 2012). This threat is called compensatory rivalry. Resentful demoralization is the opposite of the previous threat (WOHLIN et al., 2012). A subject receiving less desirable treatments may give up and not perform as good as it generally does. Depending on experiment context, participants cannot know what is the treatment and the control. So that, our platform may apply or not a treatment without participant's interference. For instance, Soares, Laureano e Borba (2002) carried out an experiment comparing traditional development against aspect-oriented development. In our platform, participants may not realize that the treatment and control are the development paradigms. They have only to know what are the tasks.

7.6 CHAPTER SUMMARY

Analyzing and evaluating software development process and source code characteristics is a major step towards achieving software product quality. The Codex is a platform modeled around a pluggable, extensible architecture that aid the execution and replication of coding experiments.

8 EVALUATING CODEX PLATFORM: PERFORMING A CODING EXPERIMENT AS A CASE STUDY

Science is facts; just as houses are made of stone, so is science made of facts; but a pile of stones is not a house, and a collection of facts is not necessarily science.

—Jules Henri Poincaré

8.1 INTRODUCTION

Chapter 6 and Chapter 7 presented the Codex metamodel and platform, respectively. They offer a set of resources and models to support the execution of experiments involving coding activities (coding experiments, for the sake of simplicity). Some pair reviews and proof of concepts provided clues about the effectiveness of Codex platform (FERREIRA et al., 2015; FERREIRA, 2014). However, it mandatory to demonstrate our platform in practice.

In this context, a case study in software engineering is an empirical inquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its natural context (RUNESON, 2003). This chapter presents and discusses the results obtained from a particular case study with our platform. Moreover, the natural context of our platform is a coding experiment, since Codex platform goal is to support coding experiment execution.

Other solutions to support SE experiments also were used in experiments (VOKÁČ et al., 2004; BRITO; NETO, 2012; SVAHNBERG; AURUM; WOHLIN, 2008). However, to the best of our knowledge, none of them treated their experiments as an exploratory case study to investigate their solution in practice. In such works, their report focused on presenting only the experiment's design, results, and analysis. However, in this chapter, our focus is to evaluate our solution, taking into account the perspective of researchers and participants, when the Codex platform is used in an experiment. The case study was carried out with researchers enrolled in Experimental Software Engineering Course at the Federal University of Pernambuco. Our primary data sources were observations and interviews with participants. Such data sources were triangulated with memos of observer-participant and experiment protocol. The case study revealed that Codex platform is a valuable resource for designing and executing coding experiments. However, it also revealed limitations while using the platform.

The rest of the chapter is composed as follows. Section 8.2 presents the research design. Section 8.3 presents some results of the case study. Section 8.4 discusses the results of our case study. Section 8.5 summarizes and offers suggestions for further research.

8.2 CASE STUDY

The primary goal of our case study is to investigate whether Codex Platform is useful to support coding experiments. As introduced before, a case study is recommended since it is a suitable way of managing an empirical investigation into a particular and current phenomenon in its natural context (ROBSON; MCCARTAN, 2016). This case study followed guidelines proposed by Runeson e Höst (2009) for conducting and reporting case study research in SE. These guidelines establish necessary process steps to carry out a case study. The following sections describe details of the activities included in each step.

8.2.1 Research Questions

This section presents the rationale for selecting each research questions. Based on the primary goal, we proposed the following general research question:

"Does an MDE approach based on coding experiment specification aid the execution of coding experiments"

To make concrete our case study scope, the general question was detailed into three more specific questions, which guided this exploratory research work:

- RQ1. Can a coding experiment be executed with Codex platform?

  Before analyzing any property of the Codex Platform, we have to ensure the platform does not prevent coding experiment execution. As said in Chapter 6, the platform is based on an extensive literature about coding experiment. However, the platform effectiveness can only be attested in a practical experiment.

- RQ2. What are the benefits and limitation of using Codex Platform to support a coding experiment?

  Given that coding experiments may be executed with the Codex Platform, the experience of carrying out it is a valuable source of information. They can be used to identify benefits and limitation when using the Codex Platform.

- RQ3. Regarding the possible issues arising during the experiment, is there any issue cased by Codex Platform?

  In general, experiments have issues (HECKMAN; SMITH, 1995; HECKMAN; SMITH, 1999). When evaluating the Codex Platform in a coding experiment, the identified issues have to be analyzed. So that, we can assess whether such issues are due to platform use or not.

8.2.2 Definition of the Case

Case study research is defined as "the in-depth study of instances of a phenomenon in its natural context and from the perspective of the participants involved in the phenomenon". We decided to carry out a case study in an academic setting, since it is easier to recruit subjects for experiments (HÖST; REGNELL; WOHLIN, 2000) and many coding experiments are conducted in such contexts. The case was performed in the *Software Engineering Experimentation Course* at the Graduate Program in the Center for Informatics/UFPE. The objective of this course is to teach how to design and carry out experiments using appropriate methods for a given problem. In this course, students have to develop their skills in planning, analyzing and, reporting experimental data. The course had 13 attendees, and it was organized in three parts: (i) The instructor lectures on concepts of designing experiments in SE; (ii) students and instructor read and discuss each chapter of Juristo e Moreno (2013); and (iii) students carry out a half semester-long experimental project, write a report, and present the results. The case targets the third part of the course.

When performing a case study, two core concepts are the case and the unit (RUNESON; HÖST, 2009). Researchers make a distinction between a case and the unit or units of analysis in that case. Yin (2013) distinguishes between holistic case studies, where the case is reviewed as a whole, and embedded case studies where multiple units of analysis are examined within a case. The current study is bounded by time, participants and place, therefore it can be viewed as a holistic case study.

8.2.3 Case Selection

Borges et al. (2015) identified several empirical studies including coding experiments. In principle, such coding experiments are potential candidates for this case study. The main constraint for selecting a case is the time-window available. The professor and the author of this thesis identified 20 candidates from the studies identified in Chapter 5. Such experiments could start in a short period and finish in about five months (the course time window). Initially, two case studies were selected as units for the case study, however, as the research progressed, it became apparent that it was not practical to plan and execute two experiments. Finally, the experiment carried out by Santos, Mendonça e Silva (2013) was selected to be replicated. Santos, Mendonça e Silva (2013) designed an experiment for investigating the concept of God Class, which is a class or methods with a high complexity. According to Fowler e Beck (1999), a *god class* is a class that tries to do much (i.e., it has many responsibilities and instance variables). This experiment was selected because:

- *It is well-documented.* The authors reported their experiment in three documents: the experiment protocol, an article (SANTOS; MENDONÇA; SILVA, 2013), and a website [1]. Most documents are in English. However, some are in Portuguese;

[1] http://wiki.dcc.ufba.br/LES/FindingGdoClassExperiment2012

- *Its artifacts are available.* As said before, the authors produced a website for experiments. In this site, the authors provided reports and all relevant artifacts used to experiment;

- *Its experiment context.* The coding activities were performed in Eclipse IDE. Besides, the authors developed an Eclipse plug-in only to collect data from the participants. With the Codex platform, this effort could be alleviated.

8.2.4 Data Collection Procedure

After executing all coding activities, each researcher that participated in of the experiment had to answer a questionnaire. This questionnaire was designed to give information about their experience in participating. The questionnaire was composed by few open-ended questions. In summary, the participants were motivated to write all good and bad experiences in designing and conducting the experiment using the Eclipse Plug-in to detect God Class, and using the Codex platform. The full list of questionnaires is provided in Appendix A. The answers of questionnaires were our primary data source. However, as said before, we used other data sources to triangulate our findings. The other data sources were:

- *Participant-observer memos.* A researcher supervised the experiment execution. The main role of this researcher was to answer any doubt about the experiment, tools, and platform. However, the researcher was forbidden from answering any questions about the coding activity execution. During the experiment execution, the research wrote down any relevant observation.

- *Experiment protocol.* As part of *Software Engineering Experimentation Course*, students had to write a document describing the experiment protocol. An experiment protocol describes the procedural method in formulating and implementing the experiment. Protocols are written whenever it is desirable to standardize a laboratory method to ensure successful replication of results by others in the same laboratory or by other laboratories. A Detailed protocol facilitates the assessment of its results through peer review.

A source of automatically-collected data were the Codex platform logs. One of our goals with the Codex platform is to provide an automatic mean to collect data. We also used this information as source of information, specially about the effort to configure and execute coding experiments.

8.2.5 Analysis Procedure

As said before, our primary data source was the questionnaire responses from researchers. However, the number of responses was too low to generate any significant statistics. The results are, however, significant when interpreted in case study's context. Given the background, a qualitative data analysis method is recommended, since the fundamental objective of the study is to derive conclusions from data, keeping a clear chain of evidence. The chain of evidence means that a reader should be able to follow the derivation of results and conclusions achieved.

The objective of the qualitative analysis is to consolidate, reduce, and interpret data obtained from various sources, and make sense of them (MERRIAM; TISDELL, 2015). It involves labeling and coding all data to identify similarities and differences to describe the phenomenon under study. We used coding techniques compiled by Seaman (1999) to code, categorize, and synthesize data.

Data analysis followed the principles of coding technique (SALDAÑA, 2015). Initially, it began with an open coding each transcript. Post-formed codes were constructed as the coding progressed. Each code were attached to their original pieces of text. Then, the codes arising were compared to codes in similar responses from other participants. From a constant systematic comparison, we grouped them into categories that represent factors affected by the use of Codex platform. As the data analysis process progressed, relationships among categories were built.

All codes and categories discovered by the coding techniques were matched against the data in experiment protocol A and observer memos. It is essential to (i) improve the finding reliability and (ii) to identify points that were not cited by the codes. Such missing points may be necessary to understand the use of Codex platform in practice.

8.2.6 Ethics

We followed the resolution 466/12 – CNS-MS of the Brazilian National Health Council that regulates research with human subjects. The company signed a Term of Authorization, and the researchers signed a Non-disclosure Agreement (covering access to sensitive information). Both documents granted the researchers access to facilities, to the participants, and to necessary documentation. They also authorized the subjects to use work hours for the interviews. We believe this formalization reduced the possibility of participants concealing information that they would consider sensitive. Before the meetings, each participant signed an Informed Consent Form that explained the overall objective and relevance of the research, guaranteeing data confidentiality, anonymity, the non-obligatory nature of the participation, and the right to withdraw from the study at any moment. All invited individuals freely agreed to participate, and no participant withdrew from the research.

8.3 RESULTS

This section presents and discusses some results from the executed case study. Firstly, we explain how the case was executed (Section 8.3.1). After, we present a description of the researchers that participated of our experiment (Section 8.3.2). Finally, Section 8.3.3 presents all codes and categories obtained our analysis.

8.3.1 Case Execution

As mentioned, the case studied was part of the *Experimental Software Engineering Course* at Center of Informatics, Federal University of Pernambuco. The experiment lasted for ten weeks. Table 29 present each week in case execution. All participants were required to be involved in a variety of activities of the experiment under study. Such activities include design, carry out, and analyze the collected data. Some of these activities were performed during the four-hour weekly class period, and some tasks were supposed to be completed as homework. Each class was held in a classroom. In this classroom, each participant had access to individual computers. The course took place during the university's fall semester and began in August 2016 and ended in December 2016.

During the first week, students were given an overview of the general experiment process. After the class, the students had to read carefully Santos, Mendonça e Silva (2013)'s work, as example of an experiment in SE. Besides, they had to identify all concepts presented in the class to get involved in experiment context.

In the following week, any doubt about experimental concepts or about the original experiment were clarified. Then, the researchers were split in groups. Each group was responsible for designing and writing part of the experiment protocol. After the class, they had to write a first version of the protocol. This preliminary protocol included information such as variables, hypothesis, and context.

In the third week, more specific characteristics of the experiment were discussed. In particular, it was discussed the experiment design and the design of each coding task. After the class, students had to update and finish the experiment design.

In the next week, the Ph.D. candidate developed an experimental model draft, based on the experiment design proposed by the students. The resulting model was discussed in class. After discussion, the model was updated. At the same time, the group of students responsible for writing the experiment execution description started to write each phase of the experiment. Besides, one participant was sampled for conducting a pilot study. The first pilot study activity evaluated the presentation about God class and Codex plug-in. Some lacks and limitations were identified and treated.

In the sixth week, the Ph.D. candidate exposed the updated version of the presentation about God class and Codex plug-in. After the presentation, each participant was asked if there was any doubt about the plug-in. Besides, it was required to each participant to

Table 29 – The derivation and execution of the Class.

Time	Course Activities		
	In-class teaching	In-class activity	Out-of-class activity
Week 1	Experiment overview	-	Reading the paper and protocol
Week 2	Clearing doubts about the experiment	Splitting the students in groups	Structuring the document according to the assigned duty and proposing a first version of experiment design
Week 3	Clearing doubts about the document	Discussing experiment design and variation from original	Finishing the experiment design
Week 4	Presentation of the Codex Model according to the protocol	Discussing the created model	First experiment execution draft
Week 5	Presentation about god class concept and support tools	Discussion about the experiment execution	Finishing the description of experiment design
Week 6	Presentation about the Codex Plugin	Discussion about the Codex Plugin	Training on Codex Plugin and warming-up with pilot experiment execution
Week 7	-	Carrying out coding activities	Starting data analysis
Week 8-9	-	Discussing data analysis	Finishing data analysis
Week 10	-	Discussing final version of the protocol	-

perform a simple experiment case at home. The participants were encouraged to perform such case to get immerse in the platform and raise possible doubts. At the same time, the experiment execution was carried out with the updated experiment model (created in the fourth week) in the pilot study. As result of the pilot execution, some issues were fixed before experimenting with all participants.

In the seventh week, students were ready to experiment. They performed the activities in the classroom, seven of them performed it on lab computers, while four of them performed it on their laptops. After the experiment, they were asked to answer the questionnaire presented in Section 8.2.4. This information is crucial to our case study.

The following weeks were dedicated to data analysis. The lecturer and the Ph.D. candidate aided the researchers in this task. Besides, they had to report the results in the

Table 30 – General participant profile.

Gender	7 males 5 females
Experience in Industry or Academia	4.11 years
Project Roles	11 programmers 3 requirement specifiers 3 project managers 2 technician Managers 2 quality specifiers
Experience	4.72 years of programming experience 3.13 years of OO programming paradigm

Table 31 – Experience participant profile

	expert	good	low	none
Java	1	7	2	1
Refactoring	0	4	6	1
God Class	0	3	3	5
Eclipse	1	6	4	0
JDeodorant	0	1	3	7

experiment documentation. As we already said in Section 1.2, this thesis aims at execution and data collection in coding experiments, therefore data analysis and discussion about results are out of case study's scope. However, it can be found in Appendix A.

8.3.2 Subject Description

As discussed in Section 8.2.2, participants were selected by convenience sampling. The sample consisted of 13 grad students enrolled in the post-grad program at CIn/UFPE. From this sample, we had one drop-out. Moreover, one participant was removed from the experiment sample to participate in the pilot experiment. Table 30 presents remaining participant profile (11 grad students) complying with ethical norms regarding anonymity. All the participants had experience in projects at Industry or Academia. Besides, the same participant could have more than one role in the same project.

Regarding participant experience, Table 31 presents the self-estimated experience to relevant characteristics. A self-estimated experience is an excellent indicator to specify the experience (LAUGWITZ; HELD; SCHREPP, 2008).

Considering Table 30 and Table 31, participants are experienced in programming and designing systems and Eclipse IDE. Moreover, they have a moderated experience in god class design. On the other hand, they had a limited knowledge on the treatment (JDeodorant). We conclude that the sample represents a good picture of the target population (people that perform coding experiment). Furthermore, the sample is not biased by the treatment.

8.3.3 Categories and Codes

Based on coding techniques (Section 8.2.5), the answer of each participant was coded. The codes were synthesized, key features of their experience were identified when participating in an experiment with the Codex platform. Figure 21 presents identified categories and sub-categories from the codes.

As Figure 21 presents, there are three main categories that synthesis all quotations made by the participants:

- **Codex Plug-in**: This category synthesis all the references made by the participant's referent to Codex Plug-in issues;
- **jDeodorant Plug-in**: This category synthesis all the references made by the participant's referent to deodorant Plug-in issues;
- **Experiment Execution**: This category synthesis all the references made by the participant's referent to experiment execution.

The next sections details each category and its subcategories.

8.3.3.1 Codex Plug-in Category

Eight out of ten participants made comments about the Codex Platform. Following an axial coding technique, we categorized each quotation in sub-categories. The identified sub-categories are:

- **Distributed vs Presential Execution groups**. As said in Section 8.3.1, the participants executed the pilot and training at their computer in their houses. However, the experiment execution was implemented at the course's laboratory. Six participants commented on a co-located and distributed execution. Table 32 presents the factors pinpoint by the participants. One participant said that a complete distributed execution would result in a catastrophe due to the bugs found in the Codex Plug-in. Two other participants discourage a distributed execution since the doubts could not be cleared up. However, three other participants commented that a distributed execution could be natural. Two participants asked if they could use our platform in experiments of their researches;

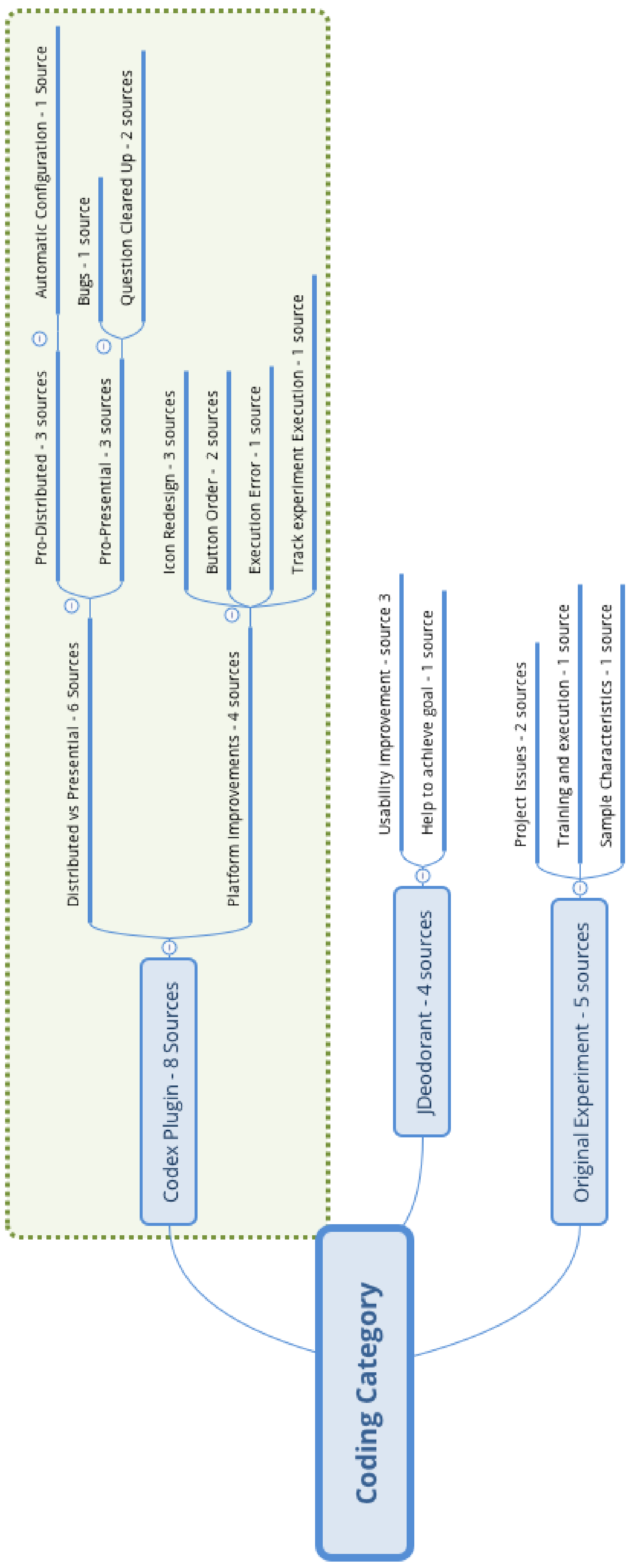

Coding Category
Codex Plugin - 8 Sources
Distributed vs Presential - 6 Sources
Pro-Distributed - 3 sources
Automatic Configuration - 1 Source
Pro-Presential - 3 sources
Bugs - 1 source
Question Cleared Up - 2 sources
Platform Improvements - 4 sources
Icon Redesign - 3 sources
Button Order - 2 sources
Execution Error - 1 source
Track experiment Execution - 1 source
JDeodorant - 4 sources
Usability Improvement - source 3
Help to achieve goal - 1 source
Original Experiment - 5 sources
Project Issues - 2 sources
Training and execution - 1 source
Sample Characteristics - 1 source

Figure 21 – Coding categories found.

Table 32 – Factor evaluation according to the participants.

Factor	Training	Experiment
Bug	-	-
Clear doubting	+	-
Automatic Configuration	+	+

- **Platform Improvements**. Four subjects related limitations in Codex Plug-in. The majority of the comments were about the tool's usability (three participants). They said that the icons and the order of buttons are not intuitive. One participant also commented about the experiment activity tracking. He said that should be a mechanism to know what is the current and next tasks.

The categories presented in this section were also mentioned in other data sources. The Co-located or Distributed Experiment Execution was referred to in observer-participant memos. As said in Section 8.3.1, the training, and pilot were executed distributed. In some memos, the researcher observed that not all participants executed the training and pilot. Besides, such participants had more doubts about the Codex plug-in. Regarding the experiment protocol, there few information about this category. The participants only mentioned that the training and pilot were executed distributed, while the experiment execution was co-located.

One participant mentioned a critical bug that occurred during the experiment. This bug was also reported in both experiment protocol and the observation report. The observer-participant mentioned that during the experiment execution, some data collected by the Codex Plug-in had was missed. Debugging the tool, we observed that a stack overflow exception was thrown. Unfortunately, it could not be observed during the pilot neither in the warm-up. Analyzing tool's logs, we concluded that it happens due to a restricted memory size of the laboratory machines. In the observer participant's memos, this bug occurrence was documented. In the experiment protocol, it can be seen indirectly. In the original version of the protocol, there was one hypothesis regarding the information that could not be collected. After the experiment execution, this assumption were removed from the protocol.

Regarding other limitations, the other sources had few data about it. The usability issues were observed by the participants that had not performed the training and warm-up. However, some participant that had performed them also mentioned some problems in the usability.

8.3.3.2 jDeodorant Category

As said in Section 8.2.3, the treatment in our experiment was the jDeodorant plug-in. Four participants commented about the jDeodorant. In fact, the aim of our experiment was to evaluate the codex platform in place. However, the subject was free to comment anything in the experiment. We did it to reduce bias in their answers.

Similar to Codex Plug-in, the majority of the comments were about the usability of the tool. Three of four participants that mentioned the jDeodorant were concerned about the usability of the plug-in. One of the participants said even that the results of the tools were not reliable. However, another subject said that the tool was usefulness to find god classes in source codes.

Regarding the other data sources, observer-participants memos mentioned that some participants had doubts with the jDeodorant. As said in Section 8.3.1, in Week 5, the participants had training with the jDeodorant tool. Unlike the Codex Platform, the training was not distributed, it was in the laboratory.

8.3.3.3 Original Experiment Category

The last category identified by the coding technique is the Experiment Execution. This category includes the codes and quotes about the way how the experiment was conducted. Five participants made seven comments about it.

Two participants made a few remarks about the size of the project. The experiment was carried out precisely as Santos, Mendonça e Silva (2013) documented it. However, according to Chatzigeorgiou e Manakos (2010), the number of bad smells increases with the size of the project. In the original experiment, only three of six scenarios had at least one god classes. Moreover, these three projects had only one or two god class. Some participants said to the participant-observed that they guessed that each project had to have at least one god class by a project. However, before the experiment execution and in the questionnaire, it is stated that some projects may not have god classes.

One remarkable comment is about the time between the training and the experiment execution. There was a gap of one week between the training and the experiment execution (Section 8.2.3). It was an extended period to ensure that the everything is fresh in the memory of the participants. We mitigated this threat, a warm-up before the experiment was performed with both the Codex plug-in and the jDeodorant.

The information cited in this category was mentioned in the experimental protocol. However, the participants mentioned it as future work. We did not find any remark about this category in the participant-observer memos.

8.4 DISCUSSION

The case study presented in this chapter explores the use of the Codex Platform in a coding experiment. The platform provided support to design and configure the development environment to perform the coding tasks. However, some issues were observed during the execution of such tasks. This information was found in the data sources and information mining of them. The rest of this section presents a discussion on the results regarding each research question based on the case study results. Besides, this section discusses limitations and validity of this results and directions for future research.

8.4.1 RQ1. Can a coding experiment be executed with the Codex Platform?

Regarding the whole case study, the coding experiment was successfully carried out supported by the Codex Platform. The platform supported the activities: coding activity specification, coding environment configuration, and data collection. However, during the experiment execution occurred some issues (Section 8.3.3.1).

The majority of the participant's quotes regards enhancements. They include improvements in the Codex Platform, the jDeodorant, and the original experiment. The next section details the found benefits and limitations (Section 8.4.2).

Comparing the effort reported by Santos, Mendonça e Silva (2013) and this replication, we observed a some differences. In the original experiment, the authors had to implement a plug-in to manage the experiment execution, and each participant had six versions of Eclipse (one for each task). Such characteristics improve the effort required to experiment. On the other hand, this replication did not require such effort since the platform could manage all these activities. With the Codex Platform, such effort is not mandatory. However, there is an extra effort to model the experiment based on the experiment protocol and artifacts.

Finally, this case study can be seem as a proof of concept. More experiments are needed for a complete evaluation of the platform. Not only other replications of the experiment presented here, but also other coding experiments in other contexts.

8.4.2 RQ2. What are the benefits and limitations in use the Codex Platform?

The codes presented in Section 8.3.3 exhibit some benefits and limitations of the Codex platform. One of them was the defects occurred in the experiment (especially, the error during the collected data serialization). This issue is critical and treats the use of the platform.

Many participant's quotes are related to usability. It is a critical issue too in the Codex Platform. A usability evaluation is required to foster the use of the platform in another experiment. This assessment can follow the guidelines to usability evaluation (TURNELL; QUEIROZ, 1996; QUEIROZ; FERREIRA, 2009). One of the participants recommends a panel presenting the performed and following activities. However, according to Wohlin et al.

(2012), such recommendation may not be beneficial. Knowing the next and quantity of tasks tend to put pressure on the participant.

Distributed and co-located execution is an ambiguous issue, there is no consensus about the realization of an experiment only in the platform, without being in a laboratory. However, it is necessary fixing all bugs before executing a distributed experiment. Besides, according to the literature on distance education (PORTER, 1997; MOORE; KEARSLEY, 2011), in a distributed execution, videos and web tutorials are recommended.

Another issue that comes from the codes in Section 8.3.3.1 is a co-located training and supervision. We encourage such practice when using the Codex Platform. It does not mean that all the participants have to experiment at the same time. However, we recommend that a supervisor shall be available during the experiment execution, especially if it is the first time with the platform. Ko, LaToza e Burnett (2015) observed the same in other tool evaluations. More experiments may show that participants confident with the platform may perform a whole distributed experiment.

8.4.3 RQ3. Comparing the original experiment and this replication, are there exclusive issues due to the use of Codex Platform?

The code presented in Section 8.3.3.3 are not due to the Codex Platform. Such code is also relevant to the original experiment. It may suggest that our platform does no influences the execution of the experiment.

Both the Codex plug-in and the treatment (jDeodorant) are Eclipse plug-ins. Section 8.3.3.2 presents many codes about the usability issues for the jDeodorant. Therefore, usability is an issue for any tool (it is not strict to the Codex plug-in or the jDeodorant). Moreover, as said in previous section (Section 8.4.2), a usability evaluation is required.

8.4.4 Addressing Limitations, Validity, and Reliability

The validity of a study denotes the trustworthiness of the results, to what extent the results are true and not biased by the researchers' subjective point of view. Threats are divided into four classes: (i) Internal, (ii) External, (iii) Construction, and (iv) Reliability (RUNESON, 2003) The next sections list identified threats and some actions to mitigate them.

8.4.4.1 Construction Validity

Selection of participants: results are dependent on the people interviewed and the lack of experience can lead to misconceptions. Aiming at mitigating this threat, we selected 12 researchers with various backgrounds. Therefore, our platform was evaluatued by diverse perspectives.

Reaction bias: there is a risk that the presence of the case study investigator influences its result. So that, obtaining favorable results due to the biased behavior. To reduce this problem, the researcher was present without commenting or interacting with any participant during the experiment execution, only observing and annotating discreetly.

Timeliness of questionnaire: questions may be misunderstood or responses misinterpreted, resulting in wrong answers. To avoid this, the questions in questionnarie were transcribed and the experimenter had to validate the answers, making corrections if they thought necessary.

8.4.4.2 Internal threats

Internal validity, or credibility, is related to the extent that the results match reality and that the researchers were able to capture reality as close as possible. We collected direct data from participants with different backgrounds. We then contrasted and compared directly collected data with observations of the observer-participant. We then used member checking to verify the completeness of our findings.

8.4.4.3 External Threats

Specific researcher: the fact that the case study occurred with only 12 researchers. It was impossible to find other researchers who fit the case selection requirements, however, to minimize this threat, the context of the selected case was explained to facilitate the understanding of how the results found here can apply to another specific context. In addition, data from various data sources were used to validate the researcher's opinion. However, it is the extent to which the results of a study can be generalized to other situations and to other people is still an issue.

8.4.4.4 Reliability

Interpretation of the data: the result can be influenced by the interpretation of the data by the researcher of the case study, inserting a bias of interpretation. To mitigate this threat, the study was designed to collect data from various sources and make it possible to triangulate these data.

8.5 CHAPTER SUMMARY

We have presented an exploratory case study for evaluating Codex platform with a potential users and practical tasks when carrying out a coding experiment. The design of the case study has been done according to the methodological framework for defining case studies presented in Runeson e Höst (2009). Although this small-scale case study will never provide general conclusions with statistical significance, the obtained results can be relevant to support other coding experiments (RIBEIRO et al., 2012; VOKÁČ et al., 2004; BRITO; NETO, 2012; SVAHNBERG; AURUM; WOHLIN, 2008). Moreover, the study was beneficial as a proof of concept to support the assumptions presented in Section 1.2. Moreover, from the Codex platform project's point of view, we have also learned: (i) the platform has shown to be useful within the context of a real coding experiment case, and (ii) the platform has the ability to automate experimental configurations within an actual coding experiment, and (iii) the platform has a potential ability to collect data from the participants automatically.

9 RELATED WORK

Talent wins games, but teamwork and intelligence win championships.

—Michael Jordan

9.1 INTRODUCTION

This chapter presents related work concerning the contributions of this thesis and the state of the art in the empirical literature. Some works are the writings that inspired and guided our research, while other studies although similar to what we performed, did not completely satisfy our goals.

9.2 MDE APPROACHES TO SUPPORT SOFTWARE ENGINEERING EXPERIMENTS

Despite the growing need for experiments in SE, carrying out such kind of empirical study is still a challenging. Furthermore, such controlled studies need to be replicated because a single controlled experiment may be insufficient and their results are limited concerning conclusions' generalization (JØRGENSEN et al., 2016). Conduction and replication of large-scale experimental SE studies are even more complicated. Concerned about the methodology of SE experiments, many researchers published procedures and guidelines aimed at improving the rigor of conducting and reporting such experiments (WOHLIN et al., 2012; JURISTO; MORENO, 2013; CARVER et al., 2014; JEDLITSCHKA; CIOLKOWSKI; PFAHL, 2008). Afterwards, some other researchers put some effort to identify and compile such relevant publications (JEDLITSCHKA; CIOLKOWSKI; PFAHL, 2008; MONTGOMERY, 2008). However, to the best of our knowledge, no effort has been directed towards cataloging and analyzing the MDE approaches to support experiments in SE.

Borges et al. (2015) surveyed all the papers published in eight well-known venues in empirical SE. Besides classifying each article according to the empirical method, that work also identified the mechanisms used to aid each empirical method. According to the authors, a mechanism is an artifice used to support the execution of the empirical method (such as tools, guidelines, and processes). However, the authors only offer a catalog of mechanisms; they do not present any comparison. Moreover, they do not focus in any particular mechanism domain.

As mentioned before, Freire et al. (FREIRE et al., 2013) performed a systematic review of the existing literature to identify the automated support tools for SE experiments. This study identified and analyzed seven tools that support the execution of experiments in

SE. However, only four of them allow to specify the experimental protocol as a model (Section 4.2). Moreover, the authors do not analyze their finding in the light of guidelines to conduct experiments in SE.

Use of modeled experiment protocol to support experimentation is commonplace in other scientific areas. For instance, the Simulation Experiment Markup Language (SED-ML) was developed to define models for biological experiment domains (WALTEMATH et al., 2011). Such models are used to validate data or replicate experiments. To better of our knowledge, there is no initiative in that direction for SE experiments.

9.3 EMPIRICAL STUDIES TO EVALUATE APPROACHES TO SUPPORT EXPERIMENTS IN SOFTWARE ENGINEERING

Freire et al. (2013) conducted a systematic review of Tool support for planning and conducting controlled experiments in general. They found out that tool support is mainly limited to the planning phase. Moreover, as presented in Chapter 4, the experimenter, is often backed up by some DSL or ontology for defining an experiment. Support means either textual as presented by Garcia et al. (2008) and Siy e Wu (2009) or graphically as of Cartaxo et al. (2012). The textual ontologies allow the formal definition of experiments and checking constraints regarding the validity of such.

Freire (2015) proposed the first integrated environment supporting all required steps of an experiment, i.e., planning, execution, and interpretation. The authors developed the ExpDSL approach for sustaining the environment. Additionally, expressibility and completeness were empirically evaluated Freire et al. (2014) in various case studies.

Various researchers also employed efforts in developing experiment environments. Mostly they differ for their application domains. In the domain of high-performance computing, Hochstein et al. (2008) developed a set of integrated tools for supporting experiments. The 'Experiment Manager Framework' is supporting almost all experiment related activities but also has some limitations. Araújo et al. (2016) proposed a similar tool; this tool focuses on experiments involving automatic execution of source code tests. However, both tools were not evaluated in other experiments.

Arisholm et al. (2002a) developed SESE (Simula Experiment Support Environment), a web-based experiment environment. Subjects are guided through the experiment with assignments. For each assignment, the time to spend it is measured. Additional material can be distributed centrally, and the framework allows the centralized collection of results and generated artifacts such as source code. The solution is applicable in many different domains introduces, however, lots of media breaks, as subjects have to switch between applications and tools. As of now, automated metric capturing for domain-specific aspects are not realizable in a web-based tool. Thus, using SESE for language related experiments is not optimal.

Similar to the SESE, the eSEE (Experimental Software Engineering Environment) has been proposed to support software engineers in their experimentation and scientific knowledge management needs (TRAVASSOS et al., 2004; LOPES; TRAVASSOS, 2010). Together with the tool proposal, the authors reported experiments performed with the eSEE. Moreover, similar to SESE, the authors published documents about the eSEE's architecture and use Brito e Neto (2012).

Considering the limitations of the previously presented approaches, Häser, Felderer e Breu (2016) have decided to provide an integrated tool environment for DSL experimentation. The authors re-used different tested languages and tools. The experiment planning was done with the ExpDSL, with its forms the base template.

The works presented in this section were evaluated in real SE experiments. However, the experiment reports focus on the experiment in detriment of how the tools support the execution of the experiment. For instance, such articles present the hypothesis and whether they were rejected or not rejected.

9.3.1 Secondary Studies to Investigate Experiments in Software Engineering

Many studies have been conducted to better understand SE experiments (BORGES et al., 2015; FREIRE et al., 2013). However, despite their contribution, such studies focused only on general aspects of SE experiments, overlooking specific dimensions of each SE sub-area, such as coding experiments. Below, we present some studies focusing on SE sub-areas.

Ko, LaToza e Burnett (2015) systematically reviewed the literature to propose a methodological guidance to design and carry out experiments that involve developers using tools. In fact, this field is similar to coding experiments, but they are not equivalent. For instance, Phongpaibul e Boehm (2006) compare code inspections and pair programming. If in one hand, it is a coding experiment, on another hand, it does not involve any tool.

Briand et al. (1999) discussed state of the art in empirical studies of object-oriented solutions. The authors present future research directions and important issues regarding the methodology of conducting such studies. However, this study only summarized the results of a working group at the Empirical Studies of Software Development and Evolution (ESSDE) workshop in May 1999. Their results are a good starting point to elucidate benefits and limitations in empirical studies involving coding experiments. However, they have a too strict scope for a more general conclusion.

Table 33 – Models for SE experiment scoping and planning phases.

	Reference	Year	Paradigm
ESEML	Cartaxo et al. (2012)	2012	DSL
ExpDSL	Freire et al. (2013)	2013	DSL
Exper Ontology	Garcia et al. (2008)	2008	Ontology
eSEE	Travassos et al. (2004)	2004	Ontology

9.4 MODELS TO SPECIFY EXPERIMENTS IN SOFTWARE ENGINEERING

We classified the related work into two categories: (i) models for SE experiments and (ii) models for software development process. The following sections present works in each category.

9.4.1 Experiment Models

We presented an extensive discussion about models that emerged to support SE experiment in Chapter 4. In this section, we only summarize our finding focusing on characteristics of interesting in our discussion. Table 33 presents the identified MDE approaches. More details are in Chapter 4.

ESEML (Empirical Software Engineering Modeling Language) is a DSL and tool to specify experiment plans in SE. The authors describe the DSL following a model also called ESEML. According to the authors, the language enables a researcher to represent all relevant information, while the tool allows an automated generation of experimental plans in PDF (CARTAXO et al., 2012).

ExpDSL (Experiment Domain-Specific Language) also comprises a DLS and tool. However, besides supporting experiment procedure specification, the tool also monitors the experiment execution. The model to specify experiments in ExpDSL has four views. The process view is responsible for defining the activities, artifacts, and roles. The metric view describes the metrics collected during the experiment execution. The experimental plan view describes information such as the factors or the statistical design. Finally, the questionnaire view represents all surveys to collect quantitative and qualitative data from participants.

ExperOntology (Experiment Ontology) is an ontology whose concepts were created to accommodate SE experiment representation. It aims to facilitate the reviewing and understanding of experimental lab packages. The ontology comprises two levels of details. The first refers to the most general controlled experiment concepts (similar to ExpDSL's experimental plan view). The second level focuses only in laboratory package. Similar to ESEML and ExpDSL, there is a tool based on this ontology to support SE experiment execution (SCATALON; GARCIA; CORREIA, 2011).

eSEE (Experimental Software Engineering Environment) is an infrastructure to manage knowledge about SE experiment definition, planning, execution, and packaging. There are two critical components in this infrastructure: the glossary and ontologies. The first aims at establishing a standard terminology in experimental SE area. The ontologies represent the formalization of the knowledge expressed in the glossary's list of terms. Moreover, this work has tool support (TRAVASSOS et al., 2004).

All approaches presented in this section focus on describing general experiment characteristics, such as variables, hypothesis, goals, etc. They do not have any mechanism to specify precisely domain-specific characteristics of coding experiments, such as artifact dependency, code, and tests. A precise specification of such characteristics is fundamental to provide automatic support for coding experiments.

9.4.2 Software Process Modeling Languages

García-Borgoñon et al. (2014) performed a systematic mapping study to identify Software Process Modeling Languages (SPML). The authors identified more than 40 languages. We highlight only languages electable to specify coding experiment characteristics:

- Business Process Modeling Notation (BPMN) is a standardized notation for creating visual models of business or organizational processes (WHITE, 2004);
- Software Process Engineering Metamodel (SPEM) is defined as a profile (UML) by the Object Management Group (SCHUPPENIES; STEINHAUER, 2002);
- Business Process Execution Language (BPEL) is an OASIS standard (ALVES et al., 2007). This language is an executable language for specifying actions within business processes with web services.

All these languages were adopted by various enterprises to specify their development process. However, regarding experiment process (as described by Wohlin et al. (2012)), these languages may be considered as a nemesis to those approaches presented in Section 9.4.1. The SPMLs are powerful enough to specify precisely all domain-specific characteristic in coding experiments. However, they lack means to specify information related to some experiment concepts, such as observed variables and treatments.

9.5 PLATFORMS TO SUPPORT EXPERIMENTS IN SOFTWARE ENGINEERING

When comparing approaches to support SE experiments against our approach, we have to consider not only the information provided in this chapter, but also information regarding our metamodel. This information is crucial, since all information modeled by our metamodel is supported by the tool described in this chapter. Moreover, this chapter describes how the coding experiment support is provided, instead of what is provided.

Controlled experiments are increasingly used to gain empirical evidence on software engineering phenomena. A systematic literature review conducted by Freire et al. (2013) reveals, however, that tool support for planning and conducting experiments is very limited. These limitations affect especially coding experiments, where conducting experiments is not only relevant for researchers but especially also for practitioners (as pointed out in Chapter 1). Most tools for experimentation are dedicated to planning phase, by supporting the experimenter with some language or ontology. That support is either graphical as proposed by Cartaxo et al. (2012) in their empirical software engineering modeling language or textual as of the approaches of Garcia et al. (2008) and Siy e Wu (2009). Both existing ontologies that allow the definition of an experiment and checking predicates or constraints regarding the validity of the experiment.

Based on their previous research Freire (2015) overcame the downsides that none of the previous approaches provided an integrated environment for all required steps of an experiment, i.e., planning, execution, and interpretation by developing their ExpDSL approach. Additionally, in contrast to the other mentioned languages and ontologies, completeness and expressibility were empirically evaluated (FREIRE et al., 2013).

On the contrary to approaches mentioned above, regarding the planning of experiments and enabling its reproducibility, efforts in developing experiment environments for different applications and domains have been taken.

Hochstein et al. (2008) developed a set of integrated tools to support experiments in software engineering for the domain of high-performance computing. The proposed Experiment Manager Framework supports quite all of the activities that are carried out in a controlled experiment. However, this framework is not tailored to allow certain adaptions for different SE domains. In particular to coding experiments, the framework cannot capture automatically data from custom-built in source codes.

A web-based experiment environment, the SESE (Simula Experiment Support Environment), was proposed by Arisholm et al. (2002b). The environment guides the subject through the assignments of an experiment, measures the time spent on each task, allows to download additional information and the centralized collection of generated artifacts such as source code. The proposed solution allows a broad range of application in very different domains. It is however not entirely suited for experiments for coding experiments, as those experiments not only rely on time measuring for executing a particular task. Source code metrics, related to the use of a specific language, may include not only the time. For automatically capturing such metrics a web-based tool is not suitable. It has to reside in the actual language workbench.

A similar work was proposed by (ARAÚJO et al., 2016). This work proposes a framework called ARREST. This framework to encourage moreover, provide reproducible experimental artifacts. An experimenter uses ARRESTT to create, share and rerun experiments with software testing techniques, such as test case selection and test suite minimisation techniques.

9.6 CHAPTER SUMMARY

The main objective of this Chapter is to provide brief information about the related work regarding each study carried out. Each related work offered guidance to our research, either explicitly described in the studies or in terms of recommendations based on the experience of the authors or through empirical evaluations results.

10 CONCLUSION

Um poema nunca se acaba; apenas se abandona.

—Paul Valery

10.1 INTRODUCTION

This chapter presents the conclusions of this research, including answers to the research questions 10.2, the main contributions (Section 10.3), the study limitations (Section 10.4), the current activities (Section 10.5), and the future works based on the gaps found in the results (Section 10.6).

10.2 ANSWERS TO THE RESEARCH QUESTIONS OF THE THESIS

This section answers the Research Questions (RQs) raised by this thesis.

RQ1: How do general-purpose platforms support experiments in software engineering?

We aimed at performing an analytical study of the currently available MDE approaches to support the experiments in SE. We have introduced a set of perspectives based on fundamental elements of main guidelines of the SE experiment. The answer to this research question can be seen in Chapter 4.

RQ2: What are the characteristics particular to coding experiments?

The goal of this research question was to map the coding experiments carried out in SE. Within this context, we synthesised the characteristics of the coding experiments in well-known venues of the empirical software engineering community. The answer to this research question can be seen in Chapter 5.

RQ3: How can the general-purpose platform approach (MDE approach) be extended to (entirely or partially) automatize some procedures in coding experiments?

Based on the results of the previous research question, we collected data from the reports of coding experiments to explore what they do when these experiments are designed, and what kinds of problems/traps they usually fall. Based on this information, we proposed a metamodel to specify coding experiment while it is compliant with the results from the RQ 1. The answer to this research question can be seen in Chapter 6.

RQ4: What are experiment facets aided by an extended MDE approach?

Considering the threats involved in coding experiments, we present how the proposed approach can mitigate these treats. We collected this threats from Wohlin et al. (2012) and other threats cited in studies analyzed when answering RQ2. The answer to this research question can be seen in Chapter 7.

RQ5: Does an MDE approach based on coding experiment specification aid the execution of coding experiments?

By answering this research question, we verified how the proposed platform supports coding experiment execution. In particular comparing the replication of a study with the platform and our replication using the platform. Moreover, we evaluated the platform from the distinct perspectives of a beginner as well as expert researchers. The result can be seen in Chapter 8.

10.3 CONTRIBUTIONS

We present the main contributions of this Ph.D. thesis as follows:

1. An Analysis of the MDE approaches. See Chapter 4 and publication (FERREIRA et al., 2017b)
2. An Analysis of the execution of coding experiments. See Chapter 5 and publication (FERREIRA et al., 2016).
3. A metamodel to describe all characteristics relevant to automate some actions when executing a coding experiment. See Chapter 6 and publication (FERREIRA et al., 2015).
4. A platform to aid the execution of coding experiments. See Chapter 7 and publication (FERREIRA, 2014; FALESSI et al., 2015).
5. Assessment of the proposed metamodel and platform. See Chapter 8.

10.4 STUDY LIMITATIONS

The following sections discuss limitations of our Ph.D. thesis research.

10.4.1 Systematic Mapping Study

The threats to validity of this research were described in Section 5.5.1in Chapter 5. In summary, the threats are bias in the papers selection and low accuracy in data extraction. The bias in the studies selection does not represent a threat to our work since we included all full paper published in the selected venues. We did not perform any automatic searches, mitigating the risk of relevant studies has not been included. Another important limitation relates to the studies not being available for download. Fortunately, we had access to all paper published in the conference proceedings.

10.4.2 Metamodel Proposition

This research has two main limitations. First, although the analysis of how experts perform their coding experiments is a strong result, the findings of this study were not empirically assessed. Second, regarding the models to specify extensive experiments in SE, we should conduct interviews with researchers from other companies and universities to try to understand their views about these models. This research can be extended to other science areas that also carry out experiments in practice.

10.4.3 Platform

The main limitation of Codex platform is lack of completeness. As stated at Chapter 4, other approaches are able to specify more information then Codex platform. However, our platform is not design to specify all relevant information about a coding experiment. This is the expereriment protocol's role. The Codex platform focus only on automating configuration and observation tasks. However, we believe that there are more tasks to me automated, then those currently supported. For instance, during the evaluation, we found many spots to improve the platform, such as configuring workspace according to code repositories (Eclipse Foundation, 2016) or DevOps (HTTERMANN, 2012).

10.4.4 Platform Evaluation

Section 8.4.4 presents in detail the threats to validity regarding the four evaluation studies. However, we highlight two of them that limit this study, namely, the small sample size and the representativeness of objects. First, although the small number of participants in experiments is a current issue in software engineering assessments, the results cannot be generalised. Therefore, it led us to consider the results only as indicators. Second, the coding experiment selected as the object to be carried out in the same experimental software engineering course, which means the characteristics of the selected coding experiment may bias the experiment. Moreover, more coding experiments need to be executed to evaluate the platform in different contexts.

10.5 CURRENT ACTIVITIES

As mentioned in previous sections, we are working towards a web-based collaborative platform for carrying out coding experiments. The proposal of this work is to provide better support to automate the execution of coding experiments, assess whether it is complete, and include all possible factors to minimise bias and maximise internal validity. This web-based platform will combine the proposed metamodel and tools to facilitate the execution of coding experiments. Also, it will permit the development of a repository of coding experiments in software engineering. This study is being developed in collaboration with the author of this thesis and a senior student of computer science at CIn UFPE.

We describe the main functionalities of the distributed collaborative platform as follows:

- All users have to access a distributed collaborative platform with a valid log-in and password. After log-in, the experiment specification and the associated executions with each user are displayed. A user can have profiles such as author, collaborator, reviewer, visitor, and administrator of the coding experiment and their activities;
- The collaborative platform have to allow the author of the coding experiment plan to include, edit, and remove information, such as "Name of the Experiment", “Name of the authors”,“Name of the reviewers”, and “Date of the coding experiment specification”. Also, the tool must provide some information, including “Initial date”, “Status that indicates where the collaboration stoped”, “the current status of each execution”, “The present percentage of the experiment execution”, and “the access to the experiment report, with the data collected from each participant”.
- The collaborative platform must allow the upload/download of the files to execute the experiment, and it must be online to facilitate the collaboration between researchers around the world.

Finally, the final version of the tool is similar to ExpDSl tool. In fact, our models are similar, therefore, implementing transformations to create a Codex model from a ExpDSL is not a challenging task. However, we plan to extend this idea to other solutions (Chapter 4), just enriching this model with any missing information.

10.6 FUTURE WORK

We present some future work that can be developed based on the research conducted in this thesis as follows:

- Updating the systematic mapping study by including more general software engineering venues to compare our results with other software engineering communities.

- Conducting a similar qualitative study with a greater number of researchers from other companies and universities to try to understand how the empirical research communities in software engineering as well as in other fields carry out experiments in practice.
- Developing the distributed collaborative platform for carrying out coding experiments using human subjects in SE based on the proposed metamodel and platform. Moreover, creating a repository of previous coding experiments for performing replication.
- Performing additional assessments using a broader range of coding experiments with different kinds of experts to confirm or not the previous results.

REFERENCES

ACCIOLY, P.; BORBA, P.; BONIFACIO, R. Comparing two black-box testing strategies for software product lines. In: IEEE. *Software Components Architectures and Reuse (SBCARS), 2012 Sixth Brazilian Symposium on*. [S.l.], 2012. p. 1–10.

ACUÑA, S. T.; GÓMEZ, M. N.; HANNAY, J. E.; JURISTO, N.; PFAHL, D. Are team personality and climate related to satisfaction and software quality? aggregating results from a twice replicated experiment. *Information and Software Technology*, Elsevier, v. 57, p. 141–156, 2015.

ALEIXO, F. A.; FREIRE, M. A.; SANTOS, W. C. dos; KULESZA, U. A model-driven approach to managing and customizing software process variabilities. In: *ICEIS (3)*. [S.l.: s.n.], 2010. p. 92–100.

ALVES, A.; ARKIN, A.; ASKARY, S.; BARRETO, C.; BLOCH, B.; CURBERA, F.; FORD, M.; GOLAND, Y.; GUZAR, A.; KARTHA, N. et al. *Web services business process execution language version 2.0 (OASIS standard). WS-BPEL TC OASIS*. 2007.

ARAÚJO, I. da C.; SILVA, W. O. da; NUNES, J. B. de S.; NETO, F. O. Arrestt: A framework to create reproducible experiments to evaluate software testing techniques. In: ACM. *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*. [S.l.], 2016. p. 1.

ARISHOLM, E.; SJØBERG, D. I. Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *Software Engineering, IEEE Transactions on*, IEEE, v. 30, n. 8, p. 521–534, 2004.

ARISHOLM, E.; SJØBERG, D. I.; CARELIUS, G. J.; LINDSJØRN, Y. Sese an experiment support environment for evaluating software engineering technologies. In: *NW-PER2002 (Tenth Nordic Workshop on Programming and Software Development Tools and Techniques). Copenhagen, Denmark*. [S.l.: s.n.], 2002. p. 81–98.

ARISHOLM, E.; SJØBERG, D. I.; CARELIUS, G. J.; LINDSJØRN, Y. A web-based support environment for software engineering experiments. *Nordic Journal of Computing*, Publishing Association Nordic Journal of Computing, v. 9, n. 3, p. 231–247, 2002.

ARISTOTLE. *Metaphysics*. The Internet Classics Archive, 350BCE. Disponível em: <http://classics.mit.edu/Aristotle/metaphysics.html>.

ASSOCIATION, A. P. et al. *Publication manual of the American psychological association*. [S.l.]: American Psychological Association Washington, 1994.

ASSOCIATION, A. P. et al. *Publication manual of the American psychological association*. [S.l.]: American Psychological Association Washington, DC, 2001.

ATKINSON, C.; KUHNE, T. Model-driven development: a metamodeling foundation. *IEEE software*, IEEE, v. 20, n. 5, p. 36–41, 2003.

AZANZA, M.; BATORY, D.; DÍAZ, O.; TRUJILLO, S. Domain-specific composition of model deltas. In: SPRINGER. *International Conference on Theory and Practice of Model Transformations*. [S.l.], 2010. p. 16–30.

BALDASSARRE, M. T.; CARVER, J.; DIESTE, O.; JURISTO, N. Replication types: Towards a shared taxonomy. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014. (EASE), p. 18:1–18:4. ISBN 978-1-4503-2476-2. Disponível em: <http://doi.acm.org/10.1145/2601248.2601299>.

BARON, R. M.; KENNY, D. A. The moderator–mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of personality and social psychology*, American Psychological Association, v. 51, n. 6, p. 1173, 1986.

BASILI, V.; GREEN, S. Software process evolution at the sel. *Ieee Software*, Ieee, v. 11, n. 4, p. 58–66, 1994.

BASILI, V. R. *Software modeling and measurement: the Goal/Question/Metric paradigm*. [S.l.], 1992.

BASILI, V. R. The experimental paradigm in software engineering. In: *Experimental Software Engineering Issues: Critical Assessment and Future Directions*. [S.l.]: Springer, 1993. p. 1–12.

BASILI, V. R.; ROMBACH, H. D. The tame project: Towards improvement-oriented software environments. *Software Engineering, IEEE Transactions on*, IEEE, v. 14, n. 6, p. 758–773, 1988.

BASILI, V. R.; SELBY, R. W.; HUTCHENS, D. H. Experimentation in software engineering. *Software Engineering, IEEE Transactions on*, IEEE, n. 7, p. 733–743, 1986.

BASILI, V. R.; SHULL, F.; LANUBILE, F. Building knowledge through families of experiments. *Software Engineering, IEEE Transactions on*, IEEE, v. 25, n. 4, p. 456–473, 1999.

BASKERVILLE, R. L.; WOOD-HARPER, A. T. A critical perspective on action research as a method for information systems research. *Journal of information Technology*, Springer, v. 11, n. 3, p. 235–246, 1996.

BENBASAT, I.; DEXTER, A. S.; MASULIS, P. S. An experimental study of the human/computer interface. *Communications of the ACM*, ACM, v. 24, n. 11, p. 752–762, 1981.

BERGERSEN, G. R.; SJØBERG, D. I. Evaluating methods and technologies in software engineering with respect to developers' skill level. In: IET. *Evaluation & Assessment in Software Engineering (EASE 2012), 16th International Conference on*. [S.l.], 2012. p. 101–110.

BERNHAUPT, R.; MIHALIC, K.; OBRIST, M. Usability evaluation methods for mobile applications. *Handbook Res User Interface Des Evaluation for Mobile Technology*, v. 44, p. 745–758, 2008.

BÉZIVIN, J.; GERBÉ, O. Towards a precise definition of the omg/mda framework. In: IEEE. *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*. [S.l.], 2001. p. 273–280.

BOEHM, B. W. et al. *Software engineering economics.* [S.l.]: Prentice-hall Englewood Cliffs (NJ), 1981. v. 197.

BORENSTEIN, N. S. Mime: a portable and robust multimedia format for internet mail. *Multimedia Systems*, Springer, v. 1, n. 1, p. 29–36, 1993.

BORGES, A.; FERREIRA, W.; BARREIROS, E.; ALMEIDA, A.; FONSECA, L.; TEIXEIRA, E.; SILVA, D.; ALENCAR, A.; SOARES, S. Support mechanisms to conduct empirical studies in software engineering. In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.* New York, NY, USA: ACM, 2014. (ESEM '14), p. 50:1–50:4.

BORGES, A.; FERREIRA, W.; BARREIROS, E.; ALMEIDA, A.; FONSECA, L.; TEIXEIRA, E.; SILVA, D.; ALENCAR, A.; SOARES, S. Support mechanisms to conduct empirical studies in software engineering: a systematic mapping study. In: ACM. *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering.* [S.l.], 2015. p. 22.

BOURQUE, P.; FAIRLEY, R. E. et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0.* [S.l.]: IEEE Computer Society Press, 2014.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, Morgan & Claypool Publishers, v. 1, n. 1, p. 1–182, 2012.

BRÉZILLON, P. Context in artificial intelligence: I. a survey of the literature. *Computers and artificial intelligence*, TRANSLIBRIS, v. 18, p. 321–340, 1999.

BRIAND, L.; ARISHOLM, E.; COUNSELL, S.; HOUDEK, F.; THÉVENOD-FOSSE, P. Empirical studies of object-oriented artifacts, methods, and processes: state of the art and future directions. *Empirical Software Engineering*, Springer, v. 4, n. 4, p. 387–404, 1999.

BRITO, J.; NETO, A. C. D. Conduzindo estudos experimentais para avaliação de uma técnica deinspeção de artefatos de teste de software. In: *12th Empirical Software Engineering Latin Amarican Workshop (ESELAW).* [S.l.: s.n.], 2012.

BUDGEN, D.; KITCHENHAM, B.; CHARTERS, S.; GIBBS, S.; POHTHONG, A.; KEUNG, J.; BRERETON, P. Lessons from conducting a distributed quasi-experiment. In: IEEE. *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on.* [S.l.], 2013. p. 143–152.

BUSE, R. P.; SADOWSKI, C.; WEIMER, W. Benefits and barriers of user evaluation in software engineering research. *ACM SIGPLAN Notices*, ACM, v. 46, n. 10, p. 643–656, 2011.

CALERO, C.; RUIZ, F.; PIATTINI, M. *Ontologies for software engineering and software technology.* [S.l.]: Springer Science & Business Media, 2006.

CAMPBELL, D. T.; STANLEY, J. C. *Experimental and quasi-experimental designs for research.* [S.l.]: Ravenio Books, 2015.

CARTAXO, B. *Mecanismos para Guiar a Caracterização de Contexto de Estudos Empíricos na Engenharia de Software: Um Mapeamento Sistemático*. Dissertação (Mestrado) — Center of Informatics - Federal University of Pernambuco, Recife - Brazil, 2014.

CARTAXO, B.; ALMEIDA, A.; BARREIROS, E.; SARAIVA, J.; FERREIRA, W.; SOARES, S. Mechanisms to characterize context of empirical studies in software engineering. In: *Experimental Software Engineering Latin American Workshop (ESELAW 2015)*. [S.l.: s.n.], 2015. p. 1–14.

CARTAXO, B.; COSTA, I.; ABRANTES, D.; SANTOS, A.; SOARES, S.; GARCIA, V. Eseml: empirical software engineering modeling language. In: ACM. *Proceedings of the 2012 workshop on Domain-specific modeling*. [S.l.], 2012. p. 55–60.

CARVER, J. C.; JUZGADO, N. J.; BALDASSARRE, M. T.; HERNÁNDEZ, S. V. Replications of software engineering experiments. *Empirical Software Engineering*, Springer, v. 19, n. 2, p. 267–276, 2014.

CATTANEO, F.; NITTO, E. D.; FUGGETTA, A.; LAVAZZA, L.; VALETTO, G. Managing software artifacts on the web with labyrinth. In: ACM. *Proceedings of the 22nd international conference on Software engineering*. [S.l.], 2000. p. 746–749.

CHATZIGEORGIOU, A.; MANAKOS, A. Investigating the evolution of bad smells in object-oriented code. In: IEEE. *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*. [S.l.], 2010. p. 106–115.

CHOI, K. S.; DEEK, F. P.; IM, I. Exploring the underlying aspects of pair programming: The impact of personality. *Information and Software Technology*, Elsevier, v. 50, n. 11, p. 1114–1126, 2008.

COMMITTEE, S. . S. E. S. et al. Ieee std 1061-1998—ieee standard for a software quality metrics methodology. *IEEE Computer Society, Tech. Rep*, 1998.

CONTE, S. D.; DUNSMORE, H. E.; SHEN, V. Y. *Software engineering metrics and models*. [S.l.]: Benjamin-Cummings Publishing Co., Inc., 1986.

COOK, T. D.; CAMPBELL, D. T.; DAY, A. *Quasi-experimentation: Design & analysis issues for field settings*. [S.l.]: Houghton Mifflin Boston, 1979. v. 351.

COSTA, I. M. do A. *The Impact of Experimentation Guides in Software Engineering: A Metanalysis*. Dissertação (Mestrado) — Center of Informatics - Federal University of Pernambuco, Recife - Brazil, 2015.

DALGAARD, P. *Introductory statistics with R*. [S.l.]: Springer Science & Business Media, 2008.

DEVOE, J. F.; PETER, K.; KAUFMAN, P.; MILLER, A.; NOONAN, M.; SNYDER, T. D.; BAUM, K. Indicators of school crime and safety, 2004. nces 2005-002. *National Center for Education Statistics*, ERIC, 2004.

DIESTE, O.; GRIMÁN, A.; JURISTO, N. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*, Springer, v. 14, n. 5, p. 513–539, 2009.

DIESTE, O.; PADUA, A. G. Developing search strategies for detecting relevant experiments for systematic reviews. In: IEEE. *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on.* [S.l.], 2007. p. 215–224.

DIJK, T. A. V. Critical discourse studies: A sociocognitive approach. *2015). Methods of Critical Discourse Studies*, p. 63–74, 2015.

DONOVAN, A.; LAUDAN, R. *Scrutinizing science: Empirical studies of scientific change.* [S.l.]: Springer Science & Business Media, 2012. v. 193.

DYBÅ, T.; KAMPENES, V. B.; SJØBERG, D. I. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, Elsevier, v. 48, n. 8, p. 745–755, 2006.

DYBÅ, T.; SJ, D. I.; CRUZES, D. S. et al. What works for whom, where, when, and why? on the role of context in empirical software engineering. In: IEEE. *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on.* [S.l.], 2012. p. 19–28.

EASTERBROOK, S.; SINGER, J.; STOREY, M. anne; DAMIAN, D. *Selecting Empirical Methods for Software Engineering Research.* 2008.

ECLIPSE, E. Eclipse modeling framework. *URL http://www. eclipse. org/modeling/emf*, 2006.

Eclipse Foundation. *Eclipse Che.* 2016. Disponível em: <http://www.eclipse.org/che/>.

EMAM, K.; CARD, D. Iso/iec15939software engineering-soft-ware measurement process. *International Standards Organization (ISO) IEC*, 2002.

FALCAO, L.; FERREIRA, W.; BORGES, A.; NEPOMUCENO, V.; SOARES, S.; BALDASSARE, M. T. An analysis of software engineering experiments using human subjects. In: IEEE. *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on.* [S.l.], 2015. p. 1–4.

FALCAO, L. C. T. *Analysis of Human-Centric Software Engineering Experiments: A Systematic Mapping Study.* Dissertação (Mestrado) — Center of Informatics - Federal University of Pernambuco, Recife - Brazil, 2016.

FALESSI, D.; CODABUX, Z.; RONG, G.; STAMELOS, I.; FERREIRA, W.; WIESE, I. S.; BARREIROS, E.; QUESADA-LOPEZ, C.; TSIRAKIDIS, P. Trends in empirical research: the report on the 2014 doctoral symposium on empirical software engineering. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 40, n. 5, p. 30–35, 2015.

FAVRE, J.-M.; NGUYEN, T. Towards a megamodel to model software evolution through transformations. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 127, n. 3, p. 59–74, 2005.

FEIGENSPAN, J.; KÄSTNER, C.; LIEBIG, J.; APEL, S.; HANENBERG, S. Measuring programming experience. In: IEEE. *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on.* [S.l.], 2012. p. 73–82.

FEITELSON, D. G. Using students as experimental subjects in software engineering research–a review and discussion of the evidence. *arXiv preprint arXiv:1512.08409*, 2015.

FENTON, N.; PFLEEGER, S. L.; GLASS, R. L. Science and substance: A challenge to software engineers. *IEEE software*, IEEE, v. 11, n. 4, p. 86–95, 1994.

FERREIRA, W. Together we are stronger: facilitating the conduction of distributed human-oriented experiments. In: ACM. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.], 2014. p. 56.

FERREIRA, W.; BALDASSARRE, M. T.; SOARES, S.; VISAGGIO, G. Toward a meta-ontology for accurate ontologies to specify domain specific experiments in software engineering. In: *Product-Focused Software Process Improvement*. [S.l.]: Springer, 2015. p. 455–470.

FERREIRA, W.; BALDASSARRE, M. T.; SOARES, S.; CARTAXO, B.; VISAGGIO, G. A comparative study of model-driven approaches for scoping and planning experiments. In: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: ACM, 2017. (EASE'17), p. 78–87. ISBN 978-1-4503-4804-1. Disponível em: <http://doi.acm.org/10.1145/3084226.3084258>.

FERREIRA, W.; BALDASSARRE, M. T.; SOARES, S.; CARTAXO, B.; VISAGGIO, G. A comparative study of model-driven approaches for scoping and planning experiments. In: ACM. *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. [S.l.], 2017. p. 78–87.

FERREIRA, W.; FALCãO, L.; BALDASSARE, T.; SOARES, S.; VISAGIO, G. Coding experiments in software engineering. Submitted. 2016.

FINK, A. *The survey handbook*. [S.l.]: Sage, 2003. v. 1.

FOUNDATION, E. Eclipse usage data collector. In: ECLIPSE. *http://www.eclipse.org/org/usagedata/*. [S.l.], 2018.

FOWLER, M. *Domain-specific languages*. [S.l.]: Pearson Education, 2010.

FOWLER, M.; BECK, K. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 1999.

FREIRE, M.; ALENCAR, D.; CAMPOS, E.; MEDEIROS, T.; KULESZA, U.; ARANHA, E.; SOARES, S. Automated support for controlled experiments in software engineering: A systematic review. *SEKE, Boston/USA*, 2013.

FREIRE, M.; KULESZA, U.; ARANHA, E.; NERY, G.; COSTA, D.; JEDLITSCHKA, A.; CAMPOS, E.; ACUÑA, S. T.; GÓMEZ, M. N. Assessing and evolving a domain specific language for formalizing software engineering experiments: An empirical study. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific, v. 24, n. 10, p. 1509–1531, 2014.

FREIRE, M. A. *Formalização de Experimentos controlados em Engenharia de Software*. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, 3 2015.

FREIRE, M. A.; ACCIOLY, P. R.; SIZÍLIO, G.; NETO, E. C.; KULESZA, U.; ARANHA, E.; BORBA, P. A model-driven approach to specifying and monitoring controlled experiments in software engineering. In: SPRINGER. *PROFES*. [S.l.], 2013. p. 65–79.

FUCCI, D.; SCANNIELLO, G.; ROMANO, S.; SHEPPERD, M.; SIGWENI, B.; UYAGUARI, F.; TURHAN, B.; JURISTO, N.; OIVO, M. An external replication on the effects of test-driven development using a multi-site blind analysis approach. In: ACM. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. [S.l.], 2016. p. 3.

FUCCI, D.; TURHAN, B. A replicated experiment on the effectiveness of test-first development. In: IEEE. *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*. [S.l.], 2013. p. 103–112.

FUCCI, D.; TURHAN, B.; JURISTO, N.; DIESTE, O.; TOSUN-MISIRLI, A.; OIVO, M. Towards an operationalization of test-driven development skills: An industrial empirical study. *Information and Software Technology*, Elsevier, v. 68, p. 82–97, 2015.

GARCÍA-BORGOÑON, L.; BARCELONA, M.; GARCÍA-GARCÍA, J.; ALBA, M.; ESCALONA, M. J. Software process modeling languages: A systematic literature review. *Information and Software Technology*, Elsevier, v. 56, n. 2, p. 103–116, 2014.

GARCIA, R. E.; HÖHN, E. N.; BARBOSA, E. F.; MALDONADO, J. C. An ontology for controlled experiments on software engineering. In: *SEKE*. [S.l.: s.n.], 2008. p. 685–690.

GEWANDTER, J. S.; MCDERMOTT, M. P.; MCKEOWN, A.; SMITH, S. M.; PAWLOWSKI, J. R.; POLI, J. J.; ROTHSTEIN, D.; WILLIAMS, M. R.; BUJANOVER, S.; FARRAR, J. T.; GILRON, I.; KATZ, N. P.; ROWBOTHAM, M. C.; TURK, D. C.; DWORKIN, R. H. Reporting of intention-to-treat analyses in recent analgesic clinical trials: {ACTTION} systematic review and recommendations. *PAIN®*, v. 155, n. 12, p. 2714 – 2719, 2014. ISSN 0304-3959.

GILB, T.; GRAHAM, D.; FINZI, S. *Software inspection*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1993.

GLASS, R. L. The software-research crisis. *IEEE Software*, IEEE, v. 11, n. 6, p. 42–47, 1994.

GLASS, R. L.; VESSEY, I.; RAMESH, V. Research in software engineering: an analysis of the literature. *Information and Software technology*, Elsevier, v. 44, n. 8, p. 491–506, 2002.

GREEN, S. B.; SALKIND, N. J. *Using SPSS for Windows and Macintosh: Analyzing and understanding data*. [S.l.]: Prentice Hall Press, 2010.

GREENFIELD, J.; SHORT, K. Software factories: assembling applications with patterns, models, frameworks and tools. In: ACM. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. [S.l.], 2003. p. 16–27.

HARRY, M. J. Six sigma: a breakthrough strategy for profitability. *Quality progress*, American Society for Quality, v. 31, n. 5, p. 60, 1998.

HÄSER, F.; FELDERER, M.; BREU, R. An integrated tool environment for experimentation in domain specific language engineering. In: ACM. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering.* [S.l.], 2016. p. 20.

HAYES, N. *Doing qualitative analysis in psychology.* [S.l.]: Psychology Press, 2013.

HAYES, W.; OVER, J. W. *The Personal Software Process (PSPSM): An Empirical Study of the Impact of PSP on Individual Engineers.* [S.l.], 1997.

HECKMAN, J. J.; SMITH, J. A. Assessing the case for social experiments. *The Journal of Economic Perspectives*, JSTOR, v. 9, n. 2, p. 85–110, 1995.

HECKMAN, J. J.; SMITH, J. A. The pre-programme earnings dip and the determinants of participation in a social programme. implications for simple programme evaluation strategies. *The Economic Journal*, Wiley Online Library, v. 109, n. 457, p. 313–348, 1999.

HOCHSTEIN, L.; NAKAMURA, T.; SHULL, F.; ZAZWORKA, N.; BASILI, V. R.; ZELKOWITZ, M. V. An environment for conducting families of software engineering experiments. *Advances in Computers*, Elsevier, v. 74, p. 175–200, 2008.

HÖST, M.; REGNELL, B.; WOHLIN, C. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, Springer, v. 5, n. 3, p. 201–214, 2000.

HOST, M.; WOHLIN, C.; THELIN, T. Experimental context classification: incentives and experience of subjects. In: IEEE. *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on.* [S.l.], 2005. p. 470–478.

HTTERMANN, M. *DevOps for developers.* [S.l.]: Apress, 2012.

HUANG, L.; HOLCOMBE, M. Empirical investigation towards the effectiveness of test first programming. *Information and Software Technology*, Elsevier, v. 51, n. 1, p. 182–194, 2009.

HYMES, D. Models of the interaction of language and social life: toward a descriptive theory. *Intercultural discourse and communication: The essential readings*, Oxford: Blackwell Publishing Ltd, p. 4–16, 2005.

IEEE Computer Society. *Software Engineering Body of Knowledge (SWEBOK).* EUA: IEEE Press, 2014. Disponível em: <http://www.swebok.org/>.

ISO, I. *IEC 15504-2: Information Technology-Process assessment-Part 2: Performing an Assessment.* [S.l.]: Geneva, Switzerland: International Organization for Standardization, 2004.

JEDLITSCHKA, A.; CIOLKOWSKI, M. Towards evidence in software engineering. In: IEEE. *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on.* [S.l.], 2004. p. 261–270.

JEDLITSCHKA, A.; CIOLKOWSKI, M.; PFAHL, D. Reporting experiments in software engineering. In: *Guide to advanced empirical software engineering.* [S.l.]: Springer, 2008. p. 201–228.

JØRGENSEN, M.; DYBÅ, T.; LIESTØL, K.; SJØBERG, D. I. Incorrect results in software engineering experiments: How to improve research practices. *Journal of Systems and Software*, Elsevier, 2015.

JØRGENSEN, M.; DYBÅ, T.; LIESTØL, K.; SJØBERG, D. I. Incorrect results in software engineering experiments: How to improve research practices. *Journal of Systems and Software*, Elsevier, v. 116, p. 133–145, 2016.

JR, F. J. F. *Survey research methods.* [S.l.]: Sage publications, 2013.

JURISTO, N.; MORENO, A. M. *Basics of software engineering experimentation.* [S.l.]: Springer Science & Business Media, 2013.

JURISTO, N.; VEGAS, S. Using differences among replications of software engineering experiments to gain knowledge. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement.* [S.l.], 2009. p. 356–366.

KAMPENES, V. B.; DYBÅ, T.; HANNAY, J. E.; SJØBERG, D. I. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, Elsevier, v. 49, n. 11, p. 1073–1086, 2007.

KAN, S. H. *Metrics and models in software quality engineering.* [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002.

KARAHASANOVIĆ, A.; LEVINE, A. K.; THOMAS, R. Comprehension strategies and difficulties in maintaining object-oriented systems: An explorative study. *Journal of Systems and Software*, Elsevier, v. 80, n. 9, p. 1541–1559, 2007.

KEELE, S. Guidelines for performing systematic literature reviews in software engineering. In: *Technical report, Ver. 2.3 EBSE Technical Report. EBSE.* [S.l.]: sn, 2007.

KERSTEN, G. E.; KERSTEN, M. A.; RAKOWSKI, W. M. Software and culture: Beyond the internationalization of the interface. *Journal of Global Information Management (JGIM)*, IGI Global, v. 10, n. 4, p. 86–101, 2002.

KIEBURTZ, R. B.; MCKINNEY, L.; BELL, J. M.; HOOK, J.; KOTOV, A.; LEWIS, J.; OLIVA, D. P.; SHEARD, T.; SMITH, I.; WALTON, L. A software engineering experiment in software component generation. In: IEEE COMPUTER SOCIETY. *Proceedings of the 18th international conference on Software engineering.* [S.l.], 1996. p. 542–552.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

KITCHENHAM, B.; AL-KHILIDAR, H.; BABAR, M. A.; 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*, Springer, v. 13, n. 1, p. 97–121, 2008.

KITCHENHAM, B.; PICKARD, L.; PFLEEGER, S. L. Case studies for method and tool evaluation. *IEEE software*, IEEE, v. 12, n. 4, p. 52–62, 1995.

KITCHENHAM, B.; SJOBERG, D. I.; DYBA, T.; BRERETON, O. P.; BUDGEN, D.; HOST, M.; RUNESON, P. Trends in the quality of human-centric software engineering experiments–a quasi-experiment. *Software Engineering, IEEE Transactions on*, IEEE, v. 39, n. 7, p. 1002–1017, 2013.

KITCHENHAM, B. A.; PFLEEGER, S. L.; PICKARD, L. M.; JONES, P. W.; HOAGLIN, D. C.; EMAM, K. E.; ROSENBERG, J. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, IEEE, v. 28, n. 8, p. 721–734, 2002.

KLEPPE, A. G.; WARMER, J. B.; BAST, W. *MDA explained: the model driven architecture: practice and promise*. [S.l.]: Addison-Wesley Professional, 2003.

KLOPPMANN, M.; KOENIG, D.; LEYMANN, F.; PFAU, G.; RICKAYZEN, A.; RIEGEN, C. von; SCHMIDT, P.; TRICKOVIC, I. Ws-bpel extension for people–bpel4people. *Joint white paper, IBM and SAP*, v. 183, p. 184, 2005.

KO, A. J.; LATOZA, T. D.; BURNETT, M. M. A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering*, Springer, v. 20, n. 1, p. 110–141, 2015.

KÜHNE, T. Matters of (meta-) modeling. *Software and Systems Modeling*, Springer, v. 5, n. 4, p. 369–385, 2006.

LAUGWITZ, B.; HELD, T.; SCHREPP, M. Construction and evaluation of a user experience questionnaire. In: SPRINGER. *Symposium of the Austrian HCI and Usability Engineering Group*. [S.l.], 2008. p. 63–76.

LAWRANCE, J.; BOGART, C.; BURNETT, M.; BELLAMY, R.; RECTOR, K.; FLEMING, S. D. How programmers debug, revisited: An information foraging theory perspective. *Software Engineering, IEEE Transactions on*, IEEE, v. 39, n. 2, p. 197–215, 2013.

LINDSAY, R. M.; EHRENBERG, A. S. The design of replicated studies. *The American Statistician*, Taylor & Francis, v. 47, n. 3, p. 217–228, 1993.

LOPES, E. C.; SCHIEL, U.; VIEIRA, V.; SALGADO, A. C. A conceptual framework for the development of applications centred on context and evidence-based practice. In: *ICEIS (3)*. [S.l.: s.n.], 2010. p. 60–69.

LOPES, V. P.; TRAVASSOS, G. H. Uma ferramenta de apoio ao planejamento de estudos experimentais em engenharia de software. In: *Proceedings of 7th Experimental Software Engineering Latin American Workshop (ESELAW 2010)*. [S.l.: s.n.], 2010. p. 40.

LUCIA, A. D.; FASANO, F.; SCANNIELLO, G.; TORTORA, G. Comparing inspection methods using controlled experiments. In: *Proc. 12th Int. Conf. Evaluation and Assessment in Software Engineering, Bari, Italy*. [S.l.: s.n.], 2008. p. 1–10.

LUCIA, A. D.; PENTA, M. D.; OLIVETO, R.; PANICHELLA, A.; PANICHELLA, S. Labeling source code with information retrieval methods: an empirical study. *Empirical Software Engineering*, Springer, v. 19, n. 5, p. 1383–1420, 2014.

LUNG, J.; ARANDA, J.; EASTERBROOK, S.; WILSON, G. On the difficulty of replicating human subjects studies in software engineering. In: IEEE. *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. [S.l.], 2008. p. 191–200.

MCCARNEY, R.; WARNER, J.; ILIFFE, S.; HASELEN, R. V.; GRIFFIN, M.; FISHER, P. The hawthorne effect: a randomised, controlled trial. *BMC medical research methodology*, BioMed Central, v. 7, n. 1, p. 30, 2007.

MCGARRY, J. *Practical software measurement: objective information for decision makers*. [S.l.]: Addison-Wesley Professional, 2002.

MEDEIROS, F.; RIBEIRO, M.; GHEYI, R.; APEL, S.; KASTNER, C.; FERREIRA, B.; CARVALHO, L.; FONSECA, B. Discipline matters: Refactoring of preprocessor directives in the# ifdef hell. *IEEE Transactions on Software Engineering*, IEEE, 2017.

MERRIAM, S. B.; TISDELL, E. J. *Qualitative research: A guide to design and implementation*. [S.l.]: John Wiley & Sons, 2015.

MONTGOMERY, D. C. *Design and analysis of experiments*. [S.l.]: John Wiley & Sons, 2008.

MOORE, M. G.; KEARSLEY, G. *Distance education: A systems view of online learning*. [S.l.]: Cengage Learning, 2011.

MOSHEIOV, G. Scheduling problems with a learning effect. *European Journal of Operational Research*, Elsevier, v. 132, n. 3, p. 687–693, 2001.

MUEHLEN, M. Z.; RECKER, J. How much language is enough? theoretical and practical use of the business process modeling notation. In: SPRINGER. *International Conference on Advanced Information Systems Engineering*. [S.l.], 2008. p. 465–479.

MÜLLER, M. M. Do programmer pairs make different mistakes than solo programmers? *Journal of Systems and Software*, Elsevier, v. 80, n. 9, p. 1460–1471, 2007.

MÜLLER, M. M.; HÖFER, A. The effect of experience on the test-driven development process. *Empirical Software Engineering*, Springer, v. 12, n. 6, p. 593–615, 2007.

MURPHY, G. C.; KERSTEN, M.; FINDLATER, L. How are java software developers using the elipse ide? *Software, IEEE*, IEEE, v. 23, n. 4, p. 76–83, 2006.

NG, T. H.; YU, Y. T.; CHEUNG, S. C.; CHAN, W. K. Human and program factors affecting the maintenance of programs with deployed design patterns. *Information and Software Technology*, Elsevier, v. 54, n. 1, p. 99–118, 2012.

Object Management Group (OMG). *UML 2.4.1 Superstructure Specification*. 2011.

OMG. *Unified Modeling Language, Infrastructure and Superstructure (Version 2.2, OMG Final Adopted Specification)*. [S.l.]: Object Management Group, 2009.

ORNE, M. T. On the social psychology of the psychological experiment: With particular reference to demand characteristics and their implications. *American psychologist*, American Psychological Association, v. 17, n. 11, p. 776, 1962.

PANČUR, M.; CIGLARIČ, M. Impact of test-driven development on productivity, code and tests: A controlled experiment. *Information and Software Technology*, Elsevier, v. 53, n. 6, p. 557–573, 2011.

PEACE, G. S. *Taguchi methods: a hands-on approach.* [S.l.]: Addison Wesley Publishing Company, 1993.

PETERSEN, K.; ALI, N. B. Identifying strategies for study selection in systematic reviews and maps. In: IEEE. *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on.* [S.l.], 2011. p. 351–354.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: *EASE.* [S.l.: s.n.], 2008. v. 8, p. 68–77.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, Elsevier, v. 64, p. 1–18, 2015.

PETTICREW, M.; ROBERTS, H. *Systematic reviews in the social sciences: A practical guide.* [S.l.]: John Wiley & Sons, 2008.

PFLEEGER, S. L.; KITCHENHAM, B. A. Principles of survey research. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 26, n. 6, 2001.

PHONGPAIBUL, M.; BOEHM, B. An empirical comparison between pair development and software inspection in thailand. In: ACM. *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering.* [S.l.], 2006. p. 85–94.

PHONGPAIBUL, M.; BOEHM, B. A replicate empirical comparison between pair development and software development with inspection. In: IEEE. *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on.* [S.l.], 2007. p. 265–274.

PIGOSKI, T. M. *Practical software maintenance: best practices for managing your software investment.* [S.l.]: John Wiley & Sons, Inc., 1996.

PLATO. *The Timaeus.* CreateSpace Independent Publishing Platform, 2012. Disponível em: <https://books.google.com.br/books?id=IUkSnwEACAAJ>.

PORTER, L. R. *Creating the virtual classroom: Distance learning with the Internet.* [S.l.]: John Wiley & Sons, Inc., 1997.

POTTS, C. Software-engineering research revisited. *IEEE software*, IEEE, v. 10, n. 5, p. 19–28, 1993.

PRECHELT, L. An empirical comparison of seven programming languages. *Computer*, IEEE, v. 33, n. 10, p. 23–29, 2000.

PRECHELT, L.; UNGER, B.; TICHY, W.; BRÃ¶SSLER, P.; VOTTA, L. A controlled experiment in maintenance comparing design patterns to simpler solutions. *IEEE Transactions on Software Engineering*, v. 27, n. 12, p. 1134–1144, 2001. ISSN 00985589.

PRESSMAN, R. S. *Software engineering: a practitioner's approach.* [S.l.]: Palgrave Macmillan, 2005.

PUNTER, T.; CIOLKOWSKI, M.; FREIMUT, B.; JOHN, I. Conducting on-line surveys in software engineering. In: IEEE. *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on.* [S.l.], 2003. p. 80–88.

QUEIROZ, J. E. R. D.; FERREIRA, D. de S. A multidimensional approach for the evaluation of mobile application user interfaces. In: *Human-Computer Interaction. New Trends*. [S.l.]: Springer, 2009. p. 242–251.

RAFI, D. M.; MOSES, K. R. K.; PETERSEN, K.; MÄNTYLÄ, M. V. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: IEEE PRESS. *Proceedings of the 7th International Workshop on Automation of Software Test.* [S.l.], 2012. p. 36–42.

RATIU, D.; MARINESCU, R.; DUCASSE, S.; GIRBA, T. Evolution-enriched detection of god classes. *Proc. of the 2nd CAVIS*, p. 3–7, 2004.

RIBEIRO, M.; QUEIROZ, F.; BORBA, P.; TOLÊDO, T.; BRABRAND, C.; SOARES, S. On the impact of feature dependencies when maintaining preprocessor-based software product lines. In: ACM. *ACM SIGPLAN Notices.* [S.l.], 2011. v. 47, n. 3, p. 23–32.

RIBEIRO, M.; TOLÊDO, T.; WINTHER, J.; BRABRAND, C.; BORBA, P. Emergo: a tool for improving maintainability of preprocessor-based product lines. In: ACM. *Proceedings of the 11th annual international conference on Aspect-oriented Software Development Companion.* [S.l.], 2012. p. 23–26.

ROBSON, C.; MCCARTAN, K. *Real world research.* [S.l.]: John Wiley & Sons, 2016.

ROOT, R. W.; DRAPER, S. Questionnaires as a software evaluation tool. In: ACM. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems.* [S.l.], 1983. p. 83–87.

ROST, J.; GLASS, R. L. *The dark side of software engineering: evil on computing projects.* [S.l.]: John Wiley & Sons, 2011.

RUNESON, P. Using students as experiment subjects–an analysis on graduate and freshmen student data. In: *Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering.–Keele University, UK.* [S.l.: s.n.], 2003. p. 95–102.

RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, Springer, v. 14, n. 2, p. 131, 2009.

SACKMAN, H.; ERIKSON, W. J.; GRANT, E. E. Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM*, ACM, v. 11, n. 1, p. 3–11, 1968.

SALDAÑA, J. *The coding manual for qualitative researchers.* [S.l.]: Sage, 2015.

SALMAN, I.; MISIRLI, A. T.; JURISTO, N. Are students representatives of professionals in software engineering experiments? In: IEEE PRESS. *Proceedings of the 37th International Conference on Software Engineering-Volume 1.* [S.l.], 2015. p. 666–676.

SANTOS, J. A.; MENDONÇA, M. G. de; SILVA, C. V. An exploratory study to investigate the impact of conceptualization in god class detection. In: ACM. *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.], 2013. p. 48–59.

SANTOS, P. S. M. dos; TRAVASSOS, G. H. esee–ambiente de apoio a experimentação em larga escala em engenharia de software. In: *1st Brazilian e-Science Workshop*. [S.l.: s.n.], 2007.

SCATALON, L. P.; GARCIA, R. E.; CORREIA, R. C. M. Packaging controlled experiments using an evolutionary approach based on ontology (s). In: *SEKE*. [S.l.: s.n.], 2011. p. 408–413.

SCHREIBER, J. B.; NORA, A.; STAGE, F. K.; BARLOW, E. A.; KING, J. Reporting structural equation modeling and confirmatory factor analysis results: A review. *The Journal of educational research*, Taylor & Francis, v. 99, n. 6, p. 323–338, 2006.

SCHULZ, K.; ALTMAN, D.; MOHER, D. et al. Consort 2010 statement: updated guidelines for reporting parallel group randomised trials. *BMC medicine*, BioMed Central Ltd, v. 8, n. 1, p. 18, 2010.

SCHUPPENIES, R.; STEINHAUER, S. Software process engineering metamodel. *OMG group, November*, Citeseer, 2002.

SEAMAN, C. B. Qualitative methods in empirical studies of software engineering. *Software Engineering, IEEE Transactions on*, IEEE, v. 25, n. 4, p. 557–572, 1999.

SEIDEWITZ, E. What models mean. *IEEE software*, IEEE, v. 20, n. 5, p. 26–32, 2003.

SELIC, B. The pragmatics of model-driven development. *IEEE software*, IEEE, v. 20, n. 5, p. 19–25, 2003.

SELIC, B. Personal reflections on automation, programming culture, and model-based software engineering. *Automated Software Engineering*, Springer, v. 15, n. 3, p. 379–391, 2008.

SIEGMUND, J.; KÄSTNER, C.; LIEBIG, J.; APEL, S.; HANENBERG, S. Measuring and modeling programming experience. *Empirical Software Engineering*, Springer, v. 19, n. 5, p. 1299–1334, 2014.

SIEGMUND, J.; SCHUMANN, J. Confounding parameters on program comprehension: a literature survey. *Empirical Software Engineering*, Springer, v. 20, n. 4, p. 1159–1192, 2015.

SILVA, A. R. da. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, Elsevier, v. 43, p. 139–155, 2015.

SILVA, F. Q. D.; SUASSUNA, M.; FRANÇA, A. C. C.; GRUBB, A. M.; GOUVEIA, T. B.; MONTEIRO, C. V.; SANTOS, I. E. dos. Replication of empirical studies in software engineering research: a systematic mapping study. *Empirical Software Engineering*, Springer, v. 19, n. 3, p. 501–557, 2014.

SILVA, M.; OLIVEIRA, T. Towards detailed software artifact specification with spemarti. In: ACM. *Proceedings of the 2011 International Conference on Software and Systems Process.* [S.l.], 2011. p. 213–217.

SIY, H.; WU, Y. An ontology to support empirical studies in software engineering. In: IEEE. *Computing, Engineering and Information, 2009. ICC'09. International Conference on.* [S.l.], 2009. p. 12–15.

SJØBERG, D. I.; ANDA, B.; ARISHOLM, E.; DYBÅ, T.; JØRGENSEN, M.; KARAHASANOVIC, A.; KOREN, E. F.; VOKÁC, M. Conducting realistic experiments in software engineering. In: IEEE. *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n.* [S.l.], 2002. p. 17–26.

SJOBERG, D. I.; DYBA, T.; JORGENSEN, M. The future of empirical methods in software engineering research. In: IEEE. *Future of Software Engineering, 2007. FOSE'07.* [S.l.], 2007. p. 358–378.

SJØBERG, D. I.; HANNAY, J. E.; HANSEN, O.; KAMPENES, V. B.; KARAHASANOVIC, A.; LIBORG, N.-K.; REKDAL, A. C. A survey of controlled experiments in software engineering. *Software Engineering, IEEE Transactions on*, IEEE, v. 31, n. 9, p. 733–753, 2005.

SMITH, B. *Beginning JSON.* 1st. ed. Berkely, CA, USA: Apress, 2015. ISBN 1484202031, 9781484202036.

SOARES, S.; LAUREANO, E.; BORBA, P. Implementing distribution and persistence aspects with aspectj. In: ACM. *ACM Sigplan Notices.* [S.l.], 2002. v. 37, n. 11, p. 174–190.

SOLEY, R. et al. Model driven architecture. *OMG white paper*, v. 308, n. 308, p. 5, 2000.

SOLINGEN, R. V.; BASILI, V.; CALDIERA, G.; ROMBACH, H. D. Goal question metric (gqm) approach. *Encyclopedia of software engineering*, Wiley Online Library, 2002.

SOMMERVILLE, I. Software engineering. international computer science series. *ed: Addison Wesley*, 2004.

STÄRK, U.; PRECHELT, L.; JOLEVSKI, I. Plat_forms 2011: finding emergent properties of web application development platforms. In: ACM. *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement.* [S.l.], 2012. p. 119–128.

STEFFE, L. P.; THOMPSON, P. W. Teaching experiment methodology: Underlying principles and essential elements. *Handbook of research design in mathematics and science education*, p. 267–306, 2000.

STEINBERG, D.; BUDINSKY, F.; MERKS, E.; PATERNOSTRO, M. *EMF: eclipse modeling framework.* [S.l.]: Pearson Education, 2008.

STOTTS, D.; WILLIAMS, L.; NAGAPPAN, N.; BAHETI, P.; JEN, D.; JACKSON, A. Virtual teaming: Experiments and experiences with distributed pair programming. In: SPRINGER. *Conference on Extreme Programming and Agile Methods.* [S.l.], 2003. p. 129–141.

STURM, A.; KRAMER, O. Evaluating the productivity of a reference-based programming approach: A controlled experiment. *Information and Software Technology*, Elsevier, v. 56, n. 10, p. 1390–1402, 2014.

SUESS, H.; SCHMIEDEK, F. Fatigue and practice effects during cognitive tasks lasting several hours. *Zeitschrift fur experimentelle Psychologie: Organ der Deutschen Gesellschaft fur Psychologie*, v. 47, n. 3, p. 162–179, 2000.

SVAHNBERG, M.; AURUM, A.; WOHLIN, C. Using students as subjects-an empirical evaluation. In: ACM. *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement.* [S.l.], 2008. p. 288–290.

THE Logic of Scientific Discovery (Routledge Classics). 2. ed. [S.l.]: Routledge, 2002. Paperback. (Routledge classics). ISBN 0415278449.

TICHY, W. F. Should computer scientists experiment more? *Computer*, IEEE, v. 31, n. 5, p. 32–40, 1998.

TICHY, W. F. Hints for reviewing empirical work in software engineering. *Empirical Software Engineering*, Springer, v. 5, n. 4, p. 309–312, 2000.

TICHY, W. F.; LUKOWICZ, P.; PRECHELT, L.; HEINZ, E. A. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, Elsevier, v. 28, n. 1, p. 9–18, 1995.

TRAVASSOS, G. H.; SANTOS, P. S. M. dos; NETO, P.; BIOLCHINI, J. An environment to support large scale experimentation in software engineering. In: IEEE. *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on.* [S.l.], 2004. p. 193–202.

TREFIL, J. S. *Encyclopedia of science and technology.* [S.l.]: Taylor & Francis, 2001.

TURNELL, M. d. F. Q.; QUEIROZ, J. E. R. de. Guidelines. an approach in the evaluation of human-computer interfaces. In: IEEE. *Systems, Man, and Cybernetics, 1996., IEEE International Conference on.* [S.l.], 1996. v. 3, p. 2090–2095.

VENTER, B. H. *Multi-language compilation.* [S.l.]: Google Patents, 2007. US Patent 7,219,338.

VOELTER, M.; BENZ, S.; DIETRICH, C.; ENGELMANN, B.; HELANDER, M.; KATS, L. C.; VISSER, E.; WACHSMUTH, G. *DSL engineering: Designing, implementing and using domain-specific languages.* [S.l.]: dslbook. org, 2013.

VOKÁČ, M.; TICHY, W.; SJØBERG, D. I.; ARISHOLM, E.; ALDRIN, M. A controlled experiment comparing the maintainability of programs designed with and without design patterns—a replication in a real programming environment. *Empirical Software Engineering*, Springer, v. 9, n. 3, p. 149–195, 2004.

VÖLTER, M.; STAHL, T.; BETTIN, J.; HAASE, A.; HELSEN, S. *Model-driven software development: technology, engineering, management.* [S.l.]: John Wiley & Sons, 2013.

WALTEMATH, D.; ADAMS, R.; BERGMANN, F. T.; HUCKA, M.; KOLPAKOV, F.; MILLER, A. K.; MORARU, I. I.; NICKERSON, D.; SAHLE, S.; SNOEP, J. L. et al. Reproducible computational biology experiments with sed-ml-the simulation experiment description markup language. *BMC systems biology*, BioMed Central, v. 5, n. 1, p. 1, 2011.

WANG, A. I.; ARISHOLM, E. The effect of task order on the maintainability of object-oriented software. *Information and Software Technology*, Elsevier, v. 51, n. 2, p. 293–305, 2009.

WEYUKER, E. J. Empirical software engineering research-the good, the bad, the ugly. In: IEEE. *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on.* [S.l.], 2011. p. 1–9.

WHITE, S. A. Introduction to bpmn. *IBM Cooperation*, v. 2, n. 0, p. 0, 2004.

WICKELGREN, W. A. Speed-accuracy tradeoff and information processing dynamics. *Acta psychologica*, Elsevier, v. 41, n. 1, p. 67–85, 1977.

WILKERSON, J. W.; JR, J. F. N.; MERCER, R. Comparing the defect reduction benefits of code inspection and test-driven development. *Software Engineering, IEEE Transactions on*, IEEE, v. 38, n. 3, p. 547–560, 2012.

WILLIAMS, B.; ONSMAN, A.; BROWN, T. Exploratory factor analysis: A five-step guide for novices. *Australasian Journal of Paramedicine*, v. 8, n. 3, 2010.

WOHLIN, C. Empirical software engineering research with industry: Top 10 challenges. In: IEEE. *Conducting Empirical Studies in Industry (CESI), 2013 1st International Workshop on.* [S.l.], 2013. p. 43–46.

WOHLIN, C.; AURUM, A. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering*, Springer, v. 20, n. 6, p. 1427–1455, 2015.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in software engineering.* [S.l.]: Springer Science & Business Media, 2012.

WOLF, W. A decade of hardware/software codesign. *Computer*, IEEE, v. 36, n. 4, p. 38–43, 2003.

WOOD, J. M. Understanding and computing cohen's kappa: A tutorial. *WebPsychEmpiricist. Web Journal at http://wpe. info/*, 2007.

WYLIE, G.; ALLPORT, A. Task switching and the measurement of "switch costs". *Psychological research*, Springer, v. 63, n. 3-4, p. 212–233, 2000.

YIN, R. K. *Case study research: Design and methods.* [S.l.]: Sage publications, 2013.

ZELKOWITZ, M. V.; WALLACE, D. R. Experimental models for validating technology. *Computer*, IEEE, v. 31, n. 5, p. 23–31, 1998.

ZIGURS, I.; BUCKLAND, B. K. A theory of task/technology fit and group support systems effectiveness. *MIS quarterly*, JSTOR, p. 313–334, 1998.

ZIMMERMANN, A.; LORENZ, A.; OPPERMANN, R. An operational definition of context. *Context*, Springer, v. 7, p. 558–571, 2007.

APPENDIX A – EXPERIMENT REPORT

This document is in Portuguese.

Pós-Graduação em Ciência da Computação

Tópicos Avançados em Linguagens de Programação 3 (IN1080)

Planejamento de Experimento

Replicação do artigo de Santos et al. (2013): An exploratory study to investigate the impact of conceptualization in god class detection

Professor: Sérgio Soares

Recife/PE

2016

1. Contextualização

Uma parte importante do desenvolvimento de sistemas é o *design* de seus componentes. É preciso definir o grau de granularidade aceitável para o desenvolvimento de classes, interfaces e hierarquias. Geralmente o *design* é específico ao problema que iremos resolver, mas é importante analisar e implementar o sistema, de tal maneira, que o impacto de mudanças seja baixo. Existem vários padrões de projeto para serem utilizados e com objetivos diferentes como, por exemplo, *Factory*, *Builder* ou *Visitor*. Problemas de projeto no desenvolvimento de software orientado a objetos podem ser analisados em diferentes perspectivas. No que diz respeito à refatoração e manutenibilidade de código, apresenta-se o conceito *code smell*, que, de acordo com Martin Fowler [1], é uma indicação superficial de um problema mais profundo no sistema. Este termo também pode se referir às más características do código, como código duplicado ou métodos longos.

Recentemente, alguns estudos foram realizados para entender os efeitos do *code smell* [3,4,5], porém ainda são poucos os estudos empíricos sobre este assunto, principalmente no que se refere ao papel humano na identificação de tais problemas. Neste contexto, Santos et al. [2] fazem um estudo para entender um tipo específico de *code smell*: *god classes* (i.e., classes que tem mais de uma responsabilidade no sistema). Mais especificamente, este estudo analisa como a conceitualização afeta a identificação de *god classes*, isto é, como desenvolvedores percebem e identificam o conceito de *god class* e quais as características que podem influenciar essa percepção.

As próximas seções descrevem uma replicação do experimento citado anteriormente para investigar se o uso da ferramenta JDeodorant reduz o tempo de identificação de *god classes.* Vale ressaltar que esta ferramenta é diferente daquela utilizada no trabalho original: a SourceMiner, que atualmente não é mais compatível com as versões mais recentes do Eclipse.

2. Objetivos do experimento

O objetivo deste artigo é apresentar o planejamento de uma replicação do experimento de Santos et al. [2] com alunos da disciplina de pós-graduação de Engenharia de Software Experimental da Universidade Federal de Pernambuco. Após a replicação, os resultados deste trabalho foram comparados com os dados obtidos no experimento original realizado por Santos et al. [2].

Nesta seção são detalhados os objetivos do experimento, assim como as das perguntas de pesquisa e, por fim, as métricas utilizadas.

2.1. Objetivos gerais

Analisar como diferentes pessoas percebem o conceito de *god class*.

Com o propósito de comparar os resultados com os obtidos por Santos et al. [2].

No que diz respeito à identificação de *god classes* em aplicações Java.

Do ponto de vista de alunos de pós-graduação de Ciência da Computação da disciplina de Engenharia de Software Experimental da Universidade Federal de Pernambuco.

No contexto do desenvolvimento da tese de doutorado de Waldemar Pires Neto.

2.2. Perguntas de Pesquisa

São consideradas quatro perguntas de pesquisa neste experimento, descritas a seguir. As perguntas selecionadas são as mesmas do trabalho original, o qual este experimento visa replicar.

- P_1: Quanto esforço é usado para identificar *god classes*?
- P_2: Qual o grau de concordância de desenvolvedores quanto a identificação de *god classes*?
- P_3: Quais características levam desenvolvedores a identificarem uma *god class*?
- P_4: Qual o grau de concordância entre desenvolvedores e o oráculo quanto a identificação de *god classes*?

2.3. Métricas

Para responder às perguntas de pesquisa, três métricas são utilizadas. Cada métrica está diretamente associada a uma ou mais perguntas de pesquisa.

- M_1 (P_1): Tempo, medido em minutos;
- M_2 (P_2 e P_4): Respostas dos formulários sobre a identificação de *god classes*, que podem ser *Sim*, *Talvez* e *Não*;
- M_3 (P_3): Resposta dos formulários sobre as razões que levaram uma classe a ser identificada como um *god class*.

A estrutura do formulário utilizado será apresentada nas próximas seções.

3. Planejamento do experimento

Nesta seção, o planejamento do experimento será discutido. Inicialmente serão definidas as unidades experimentais, os participantes do experimento, as variáveis de resposta, as variáveis independentes, os fatores e tratamentos e, por fim, as variáveis de bloqueio.

3.1. Unidades experimentais

As unidades experimentais são os seis projetos utilizados no trabalho original. Estes projetos são de pequena escala e tratam de aplicações e jogos implementados na linguagem Java; a lista completa dos projetos encontra-se na Seção 3.4.

3.2. Sujeitos do experimento

O experimento será executado por treze alunos da pós-graduação em ciência da computação da Universidade Federal de Pernambuco. Todos os alunos estão matriculados na disciplina de Engenharia de Software Experimental, oferecida no segundo semestre de 2016.

3.3. Variáveis de resposta

As variáveis de resposta a serem observadas no experimento são:

- V_1: Esforço para a identificação de *god classes*, calculado em unidade de tempo (minutos);
- V_2: Grau de concordância entre desenvolvedores sobre a identificação de *god classes*;
- V_3: Grau de concordância entre desenvolvedores sobre os motivos de uma classe ser identificada como *god class*;
- V_4: Grau de concordância entre desenvolvedores e o oráculo sobre a identificação de *god classes*.

3.4. Variáveis Independentes

- A linguagem de programação das aplicações avaliadas será Java;
- As aplicações avaliadas no experimento serão:
    - Tic Tac Toe;
    - Monopoly;
    - Chess;
    - Tetris;
    - Jackut;
    - Solitaire.
- O ambiente de execução do experimento será um laboratório de informática, do Centro de Informática da Universidade Federal de Pernambuco, no qual todas as máquinas apresentam a mesma configuração de *hardware* e *software*;
- A IDE na qual as aplicações serão avaliadas será o Eclipse Neon.

3.5. Fatores e Tratamentos

Este experimento analisará os efeitos de apenas um fator: a forma de identificação de *god classes* em uma aplicação. Os tratamentos desse fator são os seguintes:

- T_1: Utilizar o *plugin* JDeodorant para o Eclipse, que auxilia na identificação visual de *god classes*;
- T_2: Realizar a identificação de *god classes* de forma manual, sem o auxílio do *plugin* citado.

3.5.1. Descrição do Tratamento

O JDeodorant é um plugin para Eclipse com o objetivo de auxiliar na identificação de problemas relacionados ao design de sistemas como, por exemplo, *code smells*. O plugin apresenta a identificação do problema e sugere algumas maneiras de refatorações para solucioná-lo. Atualmente, JDeodorant identifica cinco categorias de *code smells*, entre elas, *Feature Envy*, *Type Checking*, *Long Method*, *God Class* e *Duplicated Code*. Esse *plugin* foi desenvolvido por um grupo de pesquisa do laboratório *Software Refactoring Lab*, no departamento de ciência da computação e engenharia de software da Universidade Concordia, no Canadá, e pelo grupo de pesquisa *Software Engineering Group* do departamento de informática aplicada, da Universidade de Macedônia, Thessaloniki, na Grécia.

3.6. Variáveis de Bloqueio

Os sujeitos do experimento possuem diferentes níveis de experiência em desenvolvimento de software e na linguagem de programação Java. Como não desejamos avaliar o efeito dessa variável nas variáveis de resposta, ela será considerada como uma variável de bloqueio.

3.7. Hipóteses

3.7.1. Hipótese Nula (H_0)

Não há diferença de esforços e graus de concordância entre a utilização do *plugin* no Eclipse e a identificação manual de *god classes*.

- $H0_1$: $V_1(T_1) = V_1(T_2)$
- $H0_2$: $V_2(T_1) = V_2(T_2)$
- $H0_3$: $V_3(T_1) = V_3(T_2)$
- $H0_4$: $V_4(T_1) = V_4(T_2)$

3.7.2. Hipótese Alternativa (H_1)

A utilização do plugin no Eclipse reduz os esforços e aumenta os graus de concordância, quando comparado com a identificação manual de *god classes*.

- $H1_1$: $V_1(T_1) < V_1(T_2)$
- $H1_2$: $V_2(T_1) < V_2(T_2)$
- $H1_3$: $V_3(T_1) > V_3(T_2)$
- $H1_4$: $V_4(T_1) > V_4(T_2)$

4. Desenho Experimental

Tendo em vista que os participantes do experimento são estudantes e suas respectivas experiências na indústria podem influenciar os resultados do experimento, o desenho experimental aqui descrito foi adotado.

O experimento foi realizado no Grad3, um dos laboratórios do Centro de Informática da Universidade Federal de Pernambuco. O tempo alocado para a execução foi de 1,5 horas. Dos treze alunos matriculados na disciplina, um deles participou do experimento piloto, ficando os outros doze alunos para a execução do experimento.

Foram criados dois grupos, no qual os participantes foram distribuídos de forma aleatória, visando remover o efeito da experiência individual de cada um. A Equipe 1 e a Equipe 2 ficaram com 6 participantes, cada. Todos os participantes utilizaram uma máquina para realizar o experimento, sendo que todas as máquinas estavam configuradas igualmente de forma a viabilizar o processo.

Todas as máquinas possuíam duas instâncias do Eclipse (I1 e I2): a instância 1 possuía o plugin JDeodorant e a instância I2 não possuía o plugin instalado. Para facilitar o andamento do experimento, as diferentes aplicações foram divididas em 2 grupos:

Programas G1: Monopoly, Tetris e Tic Tac Toe.

Programas G2: Chess, Jackut e Solitaire-FreeCell.

Na execução do experimento, primeiramente a equipe 1 realizou o experimento nos programas G1 utilizando a instância 1 e a equipe 2 realizou o experimento nos programas G1 na instância 2. Posteriormente, a equipe 1 realizou o experimento nos programas G2 na I2, e a equipe 2 experimentou os programas G2 na I1, como relacionado no quadro abaixo.

	G1 Programs	**G2 Programs**
Equipe 1	I1 (com plugin)	I2 (sem plugin)
Equipe 2	I2 (sem plugin)	I1 (com plugin)

Durante a realização do experimento, os participantes preencheram o formulário relativo à identificação de *god classes*. Os formulários possuem uma estrutura padrão para todos os projetos, requisitando a posição do participante quanto ao status das classes (se é ou não uma *god class*) e, se for uma *god class*, que motivos levaram a esta conclusão. O formulário também exige que o participante insira a hora de início e fim da avaliação de cada projeto. Um exemplo de formulário pode ser encontrado no Apêndice A.

5. Preparação

Todo o experimento foi realizado em dois dias, nos quais ocorreram o treinamento das ferramentas utilizadas e a execução do experimento. No primeiro dia houve uma apresentação sobre *code smell*, focada nos conceitos de *god class*, além de um treinamento do plugin JDeodorant. E, por fim, no segundo dia houve a execução do experimento a partir das instruções dadas previamente pelo instrutor.

5.1. Cronograma

Data	**Evento**
02 de Novembro de 2016	Preparação dos instrumentos
09 de Novembro de 2016	Execução do piloto
10 de Novembro de 2016	Treinamento
24 de Novembro de 2016	Execução do experimento
08 de Dezembro de 2016	Resultado final

5.2. Piloto

O instrutor Waldemar Pires Neto preparou a primeira versão da apresentação de slides para todos os participantes do experimento. Logo após a apresentação, houve um treinamento sobre a utilização do plugin JDeodorant. O treinamento consistiu na apresentação de slides e um simples exemplo prático.

Além da apresentação no piloto testamos os arquivos usados no na execução do experimento. Os bugs identificados foram corrigidos para a execução final do experimento.

5.3. Treinamento

O instrutor Waldemar Pires Neto preparou uma apresentação de slides para todos os participantes do experimento. A apresentação aconteceu de forma breve, totalizando 15 minutos, no laboratório Grad3 do Centro de Informática (UFPE). Logo após a apresentação, houve um treinamento sobre a utilização do plugin JDeodorant. O treinamento consistiu na apresentação de slides e um simples exemplo prático. Em seguida, o instrutor relatou uma sucinta explicação sobre o desenho do experimento.

Dois dias antes do experimento, o instrutor enviou para os emails dos participantes um tutorial com o objetivo de prepará-los para o experimento. Este preparatório foi realizado remotamente, e cada participante utilizou suas respectivas máquinas para fazer as instalações necessárias para o aquecimento.

5.4. Preparação dos Instrumentos

Com o objetivo de preparar o ambiente para conduzir o experimento da melhor forma, as máquinas utilizadas teriam que estar configuradas do mesmo modo. Todas as máquinas estavam localizadas no laboratório Grad 3 do Centro de Informática (UFPE) e possuíam as mesmas configurações: Sistema Operacional Windows 7 (64 bits), 8 GB de memória RAM e processador AMD Phenom II X4 B97. Porém, algumas modificações ainda precisavam ser feitas para a execução, então foram instaladas nas máquinas duas instâncias do Eclipse, na qual a primeira instância apresentava a instalação do plugin JDeodorant (versão 5.0.62) e a segunda instância não possuía o plugin. Após as devidas instalações, o ambiente estava preparado para a execução do experimento.

5.5. Execução do Experimento

O instrutor separou os doze participantes de forma aleatória a partir de um sorteio e, no final, foram formadas duas equipes. Um dos participantes, porém, não pode comparecer. Logo após a divisão das equipes, os participantes iniciaram a execução do experimento de acordo com o desenho experimental apresentado na fase de treinamento.

6. Análise dos Dados

Esta seção apresenta a análise dos dados obtidos com a execução do experimento.

P_1: Quanto esforço é usado para identificar *god classes*?

A avaliação do esforço foi feita seguindo uma metodologia parecida à de Santos et al. [2], com a diferença de que apenas o tempo marcado no formulário pelo participante durante o experimento será considerado (*quest_time*).

Os tempos individuais para o *quest_time* estão expostos nos gráficos 1 e 2. Como temos 11 participantes, e cada um desses avaliou todos os 6 projetos, temos 66 valores de tempo. Nestes, 33 estavam com uso do jDeodorant e 33 estavam sem jDeodorant .

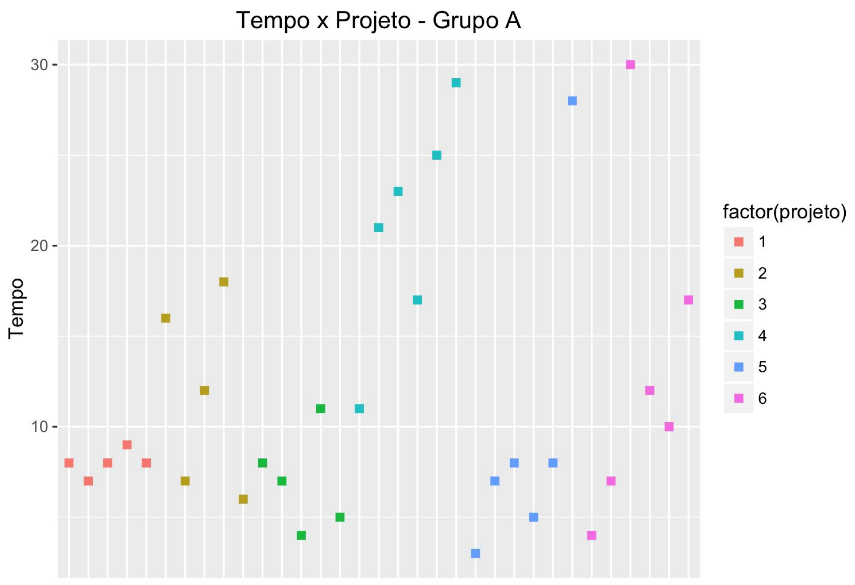


Gráfico 1. Distribuição individual do tempo por projeto para o Grupo A.

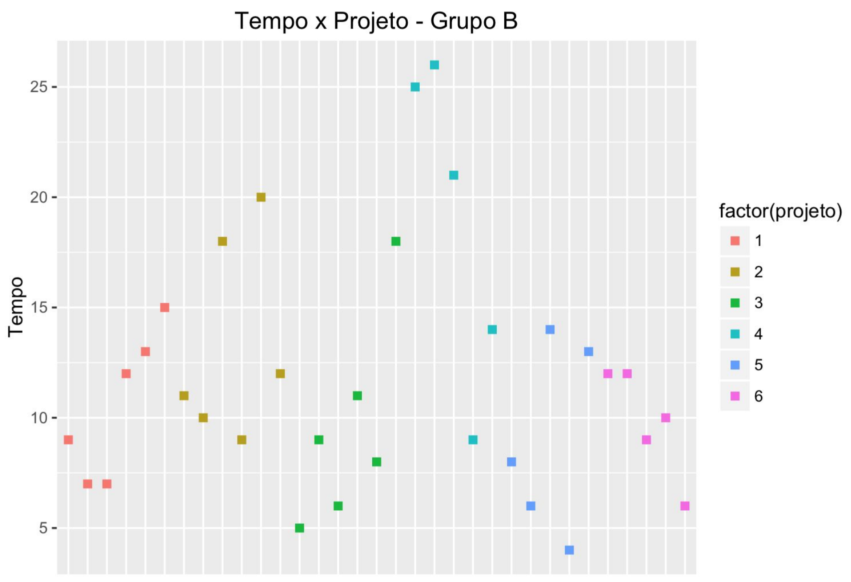


Gráfico 2. Distribuição individual do tempo por projeto para o Grupo B.

É possível notar que os projetos 1, 3 e 5 aparentam ter uma menor dispersão quanto ao tempo entre os integrantes, para ambos os grupos. Já o projeto 4 aparenta ser o que gerou as maiores variações de esforço. Um dos fatores que podem influenciar nessa variação é a diferença na quantidade de classes dos projetos. Essas afirmações podem ser vistas mais claramente nos seus respectivos boxplots (gráficos 3 e 4).

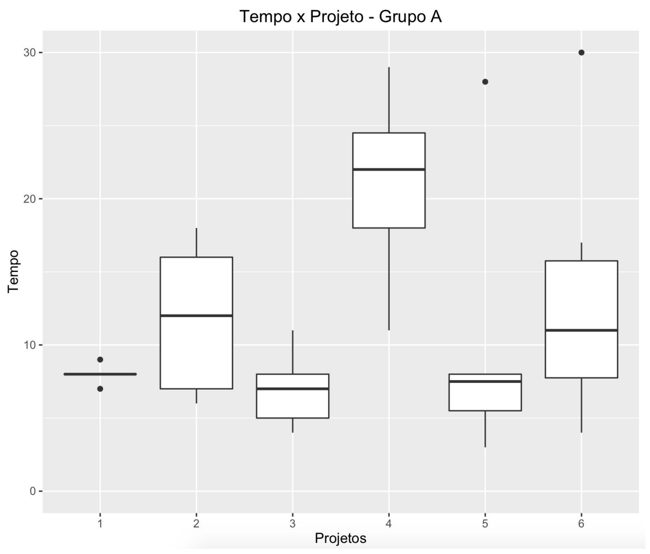


Gráfico 3. Boxplot do tempo por projeto para o Grupo A.

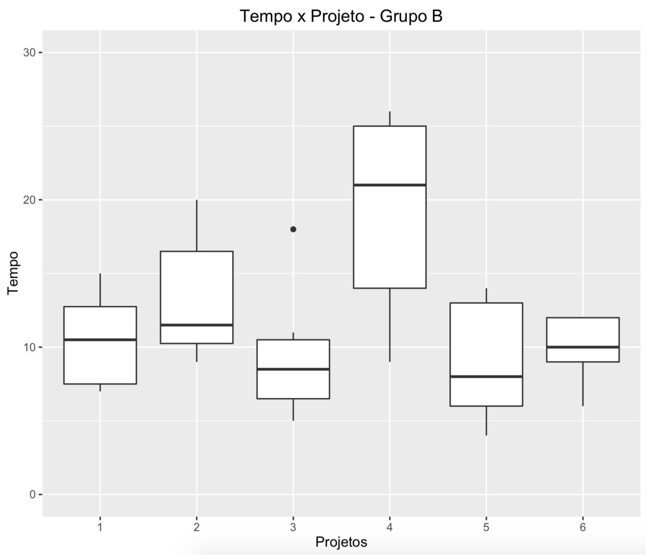


Gráfico 4. Boxplot do tempo por projeto para o Grupo B.

As distribuições de tempo dos grupos A e B aparentam ser similares, quando analisadas projeto a projeto. Com valores de mediana parecidos, os boxplots indicam que os grupos possuem esforço similar. Essa similaridade para ambos os grupos fica mais explícito no gráfico 5.

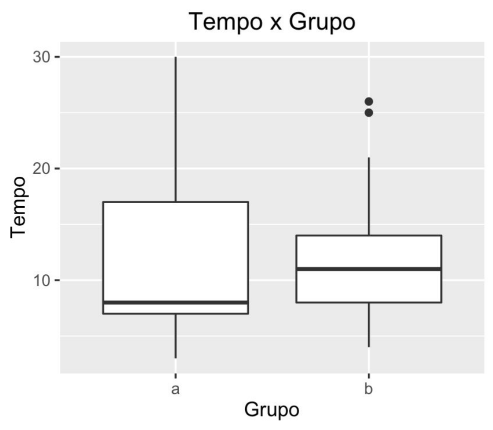


Gráfico 5. Comparação dos boxplots por grupo.

Buscando verificar se existe diferença relevante no esforço empregado entre os grupos A e B, aplicamos o Teste de Wilcoxon para comparação de médias de distribuições não-paramétricas [$H0_1$: $V_1(T_1) = V_1(T_2)$]. Esse teste foi escolhido porque os dados de tempo dos grupos A e B não seguem uma distribuição normal, de acordo com o Teste de Shapiro-Wilk de normalidade (p-value-A = 0.0003651 e p-value-B = 0.01124).

Aplicando o teste de Wilcoxon, de forma bilateral, obtivemos p-value = 0.4636. Esse valor não é suficiente para rejeitar $H0_1$. Logo, podemos afirmar que não há diferença de esforços de tempo entre a utilização do *plugin* no Eclipse e a identificação manual de *god classes*.

P_2: Qual o grau de concordância dos participantes quanto a identificação de *god classes*?

A etapa de análise de concordância em que os participantes identificaram candidatas a *god classes* foi realizada utilizando o método de Joseph Fleiss [8], também utilizado no experimento realizado por Santos et al. [2]. Este método tem como objetivo verificar o coeficiente de concordância das respostas para um determinado teste composto por *n* sujeitos, *m* avaliadores e *k* categorias.

A análise do *Fleiss Kappa* foi divida em duas etapas. Na primeira, foram coletadas as quantidades de *god classes* identificadas pelos participantes do experimento tanto do grupo A, quanto do grupo B, considerando as respostas dos campos de "*Yes*" e "*Yes or Maybe*" contidos no formulário. Em seguida, os dados obtidos através desta coleta foram catalogados, sendo separados por grupos, conforme podem ser observados descritos na Tabela 6.1 e na Tabela 6.2 distribuídas abaixo:

Tabela 6.1: Quantidade de God Classes identificadas pelos participantes do Grupo A

Grupo A						
	Monopoly	Tic Tac Toe	Tetris	Chess	Solitaire	Jackut
PA1	1	0	0	4	1	3
PA2	3	2	1	2	4	1
PA3	3	2	3	1	3	0
PA4	4	2	3	1	1	1
PA5	2	0	0	2	2	1
PA6	6	5	6			

Tabela 6.2: Quantidade de God Classes identificadas pelos participantes do Grupo B

Grupo B						
	Monopoly	Tic Tac Toe	Tetris	Chess	Solitaire	Jackut
PB1	1	1	1	1	1	0
PB2	4	0	1	2	2	2
PB3	3	1	1	1	2	1
PB4	1	0	1	4	0	3
PB5	4	1	0	0	0	2
PB6				6	5	6

Observando os dados acima é possível perceber que os campos do Participante B6 e Participante A6 estão incompletos. Para que se tornasse possível realizar a análise *Fleiss Kappa* os campos com as respostas ausentes destes participantes foram preenchidos com o valor "0" (zero). Com ambas as tabelas dispostas com todos os dados, se deu início a segunda etapa em que foram calculados os coeficientes para cada grupo de maneira separada.

Cada grupo (Grupo A e Grupo B) está composto por 6 sujeitos (projetos), localizados na primeira linha, 6 avaliadores localizados na primeira coluna e duas categorias. A primeira categoria adotada na análise consiste das respostas sugeridas por "*Yes*" e "*Yes or Maybe*" (quantidades) e a segunda categoria atribuída foi a "*None*", formada pelos espaços vazios que tiverem de ser preenchidos com 0. Após a importação dos dados ter sido realizada na ferramenta estatística R, utilizando o pacote irr (*Interrater Reliability and Agreement*), foram obtidos os seguintes resultados, descritos na Tabela 6.3 e na Tabela 6.4:

Tabela 6.4: Análise de concordância *Fleiss Kappa* para o grupo A

Fleiss' Kappa for m Raters	
Subjects	6
Raters	6
Kappa	0.0144
Z=	0.292
p-value	0.77
Interpretação	Concordância leve

Tabela 6.3: Análise de concordância *Fleiss Kappa* para o grupo B

Fleiss' Kappa for m Raters	
Subjects	6
Raters	6
Kappa	0.0927
Z=	1.7
p-value	0.0893
Interpretação	Concordância leve

Os coeficientes *Kappa* obtidos pelos Grupo A (*Kappa* = 0.0144) e Grupo B (*Kappa* = 0.0927) tiveram uma diferença pequena ($Kappa_B$- $Kappa_A$ = 0.0749) em relação ao uso do jDeodorant para a identificação das *god classes*. As análises replicam que o efeito causado com o uso da ferramenta, quando comparados o Grupo A e o Grupo B, é pequeno considerando as atividades desempenhadas pelos participantes nos testes que foram realizados.

Outro fator importante a ser constatado é a interpretação dos resultados de cada grupo. Os valores obtidos nos testes obtiveram na escala *Fleiss Kappa* uma *concordância leve*, o que significa dizer que os coeficientes têm um grau pequeno, mas que que não se torna suficiente para para rejeitar $H0_2$. Portanto, pode-se concluir que não há diferença nos esforços de quantidade entre a utilização do *plugin* na IDE Eclipse e a perspectiva de buscar manualmente as *god classes*.

P_3: Quais características levam os participantes a identificarem uma *god class*?

Para saber quais características levavam um participante a achar que certa classe era uma *God Class,* foi pedido que ele assinalasse uma ou mais opções, para cada *God Class* identificada, de acordo com a lista abaixo. O participante também estava livre para determinar seu próprio motivo, através do campo aberto "Other".

As opções disponíveis eram:

M1 - Class is not used anymore.

M2 - Class is highly complex.

M3 - Class is misplaced.

M4 - Class-Method lacks comments.

M5 - Method is wrongly named.

M6 - Method is wrongly named.

M7 - Method is highly complex.

M8 - Method is misplaced.

M9 - Attribute is not used.

M10 - Class is special (e. g. a necessary framework class).

M11 - Other.

Os motivos escolhidos por cada participante para a razão de identificar uma classe como *God Class* (tanto para *Sim* quanto para *Talvez*) se encontram na tabela abaixo.

	Motivos										
Participantes	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
P1			2				2				7
P2	1	5.8		1	0.3		1	1			1.8
P3		3	0.5				3.5	3			
P4		3.8	0.3					0.8			
P5		1					3	4		2	
P6		1.5					1.5				
P7		3.85	2.15				4.55	2.45	0.2		2.6
P8		5	0.5				0.5	1			5

P9		10		2				1.5			2.5
P10		0.5		1.5		0.5		1.5			
P11	0.55	5.25	2.2	2.95	0.95	1.9	3.35	1.95	2.95	4.45	
Total	1.55	39.7	7.65	7.45	1.25	2.4	19.4	17.2	3.15	6.45	18.9

Para evitar que ocorresse um *bias* quando algum participante escolhesse múltiplos motivos para uma mesma classe, o peso dos motivos foi diminuído quando ocorrem múltiplas seleções. Por exemplo, se uma classe recebe apenas um motivo M1, M1 então ganha *score* 1. Entretanto, se uma outra classe fosse classificada pelos motivos M2 e M3, cada um dos motivos recebeu 0.5.

O motivo mais usado, com *score* de 39.7, foi o "M2 - Class is highly complex". Essa opção foi escolhida por 10 dos 11 participantes. Logo em seguida, as opções mais escolhidas foram "M7 - Method is highly complex" (*score* 19.4), "M11 - Other" (*score* 18.9) e "M8 - Method is misplaced" (*score* 17.2).

Comparando com o experimento de Santos *et al.* [2], os três motivos mais escolhidos são os mesmos. Eles também encontraram a mesma frequência, sendo a complexidade da classe o motivo mais escolhido, seguido da complexidade dos métodos da classe. A replicação, então, confirmam as razões principais do experimento original.

P_4: Qual o grau de concordância entre participantes e o oráculo quanto a identificação de *god classes*?

Na quarta análise foram realizados os testes de concordância para identificação de *god classes* com o uso de oráculo dos Grupo A e Grupo B. Seguindo a mesma perspectiva de Santos et al. [2] foi utilizado o método *Finn Test* [9] que possibilita encontrar o *Finn Coefficient* através de uma matriz *ratings*, composta por *n* sujeitos e *m raters*, pelo *s.levels*, composto pelo número de diferentes categorias de avaliação, e pelo *model*, que estabelece a ordem de leitura e comparação de dados.

O objetivo do teste é verificar em uma amostra *n* de classes avaliadas por cada participante quais delas replicam um coeficiente de confiabilidade para serem consideradas *god classes*. Para que este objetivo fosse possível de ser alcançado foi necessário inicialmente coletar e identificar as classes sugeridas nos formulários e em seguida classificá-las por projeto.

Na classificação, descrita nas Tabela 6.5 e Tabela 6.6, cada participante foi distribuído considerando em seu grupos seguindo como regra os projetos que foram avaliados com o uso e sem o uso do *plugin*. Além disto, cada um deles ficou responsável por avaliar 3 projetos, conforme pode ser observado a seguir.

Tabela 6.5: Distribuição das god classes identificadas pelos participantes do Grupo A

Grupo A							
	Monopoly	Tic Tac Toe	Tetris		Chess	Solitary	Jackut
P1A	monopoly.Jogo (Maybe)	-	-	P7A	Bishop; King (Maybe); Queen (Maybe); Rook (Maybe)	Baralho	FacadeJackut (Maybe); Jackut (Maybe); Usuario
P2A	easy.TestFacade (Maybe); UserStoriesFacade (Maybe); Comandos (Maybe); Jogador (Maybe); Jogo; Tabuleiro	model. JogoVelha; view. JogoVelhaGui	view. tetrisGUI	P8A	Board (Maybe); ChessGUI (Maybe)	InterfacePaciencia (Maybe); ControladorGlobal (Maybe); FrameFreeCell (Maybe); JanelaSobre (Maybe)	UsuarioController (maybe)
P3A	UserStoriesFacade; Jogo; Tabuleiro	Jogador; view. JogoVelhaGui (Maybe)	model. Pecas; model. Tetris; TetrisGUI (Maybe)	P9A	Chess	ControladorGlobal; FrameFreeCell (Maybe); InterfacePaciencia (Maybe)	-
P4A	UserStoriesFacade; Banco (Maybe); Jogador; Jogo	model. JogoVelha (Maybe); view. JogoVelhaGui	model. Pecas (Maybe); model. Tetris (Maybe); TetrisGUI	P10A	Chess	ControladorGlobal (Maybe)	FacadeJackut (Maybe)
P5A	UserStoriesFacade; Jogo (Maybe)	-	-	P11A	King (Maybe); BoardGUI (Maybe)	ControladorGlobal; FrameFreeCell	Usuario

Tabela 6.6: Distribuição das god classes identificadas pelos participantes do Grupo B

	Grupo B						
	Monopoly	Tic Tac Toe	Tetris		Chess	Solitary	Jackut
P7B	Tabuleiro (Maybe)	JogoVelhaGUI (Maybe)	Peca	P1B	model.Chess (Maybe)	gui. FrameFreeCell	-
P8B	Jogo (Maybe); Jogador (Maybe); Banco (Maybe); Tabuleiro (Maybe)	-	TetrisGUI (Maybe)	P2B	pecas.King (Maybe); model.Chess	InterfacePaciencia (Maybe)	UsuarioController; FacedeJackut (Maybe)
P9B	Jogo; Jogador (Maybe); Tabuleiro (Maybe)	JogoVelha	Tetris (maybe)	P3B	model.Chess	EstruturaBaralho (Maybe); FrameFreeCell	model.Usuario

P10B	Jogo	-	Tetris (Maybe)	P4B	pecas.King (Maybe); model.Chess; pecas.Queen (Maybe); pecas.Rock (Maybe)	-	Usuario Controller; FacedeJackut; model.Usuario
P11B	Banco; Jogador; Lugar; Tabuleiro	JogoVelhaGUI (Maybe)	-	P5B	-	-	UsuarioController (Maybe); FacedeJackut
				P6B	Board; Chess (Maybe); Piece; Player; Position; PlayerPieceThreat (Maybe)	Baralho (Maybe); Main; ControladorGlobal;GlobalConfiguration(Maybe);RequestGlobal	FacadeJackut; EasyTest(Maybe); Comando (Maybe); Jackut; Mensagem; Perfil (Maybe)

Observando as Tabela 6.5 e Tabela 6.6 é possível notar que alguns participantes (P1A, P5A, P9A, P4B e P5B) ficaram com os campos dos formulários vazios na avaliação dos projetos. Diante desta condição apresentada na coleta, a montagem do oráculo utilizado por Santos et. al. [2] foi realizada neste experimento com a atribuição das categorias de avaliação para as classes, seguindo o padrão das respostas de "*Yes*" ou "*Yes or Maybe*" com peso 1 e das respostas das "*classes não citadas*" ou "*campos vazios*" com peso 0.

Para cada grupo de 3 projetos, nos Grupo A e Grupo B, foram montados dois oráculos diferentes, conforme pode ser visualizado nos exemplos das Tabela 6.7 e Tabela 6.8. Nos oráculos são contabilizados o total de *subjects* (Nclasses = total de classes a serem avaliadas) por cada projeto e os *raters* (Or1 e Or2 = respostas Yes, Maybe ou não citadas) que equivalem as respostas informadas por cada participante avaliado.

Tabela 6.7: Oráculo para avaliação dos projetos Monopoly, Tic Tac Toe e Tetris

Program	GodClass	Oracle	
		Or1	Or2
Monopoly (10 classes)			
	Jogo	-	Maybe
Tic Tac Toe (5 classes)			
	-		
Tetris (16 classes)			
	-	-	-

Tabela 6.8: Oráculo para avaliação dos projetos Chess, Solitaire e Jackut

Program		GodClass	Oracle	
			Or1	Or2
Chess (15 classes)				
		Jogo	-	Maybe
Solitaire (23 classes)				
		-		
Jackut (19 classes)				
		-	-	-

Após a realização dos testes na ferramenta estatística R, utilizando o pacote irr (*Interrater Reliability and Agreement*) foram obtidos os coeficientes *Finn* separados pelos Grupo A e Grupo B. Ao todo puderam ser comparados 11 participantes de cada grupo, totalizando 22 análises individuais conforme expressam os resultados descritos nas Tabela 6.9 e Tabela 6.10.

Tabela 6.9: Coeficiente *Finn* com o uso do Oráculo do Grupo A

Participante	Finn
P1A	0.935
P2A	0.419
P3A	0.484
P4A	0.419
P5A	0.871
P6A	0.0968 (Negativo)
P7A	0.719
P8A	0.754
P9A	0.86
P10A	0.895
P11A	0.825

Tabela 6.10: Coeficiente *Finn* com o uso do Oráculo do Grupo B

Participante	Finn
P1B	0.93
P2B	0.825
P3B	0.86
P4B	0.754
P5B	0.93
P6B	0.404
P7B	0.806
P8B	0.677
P9B	0.677
P10B	0.871
P11B	0.677

Comparando-se os resultados obtidos nos testes percebe-se que os índices dos coeficientes tiveram valores que variaram entre -0.0968 (P6A) e 0.935 (P1A) para o Grupo A e entre 0.404 (P6B) e 0.93 (P1B e P5B). A diferença entre estes coeficientes para o Grupo A foi de 0.8382 e para o Grupo B de 0.531, o que significa dizer que o Grupo B obteve uma concordância maior nas classes identificadas como *god classes* quando comparada ao Grupo A.

Também é possível verificar que a maioria dos coeficientes dos grupos se aproximam de 1 (Concordância total), representando uma confiabilidade relevante dos avaliadores. Apesar de 1 dos 22 participantes ter obtido um valor negativo -0.0968 (P6A) (Discordância total) este fator não se torna suficiente para rejeitar a $H0_4$. Portanto, pode-se concluir que não há diferença de esforços e graus de concordância entre a utilização do *plugin* no Eclipse e a identificação manual de *god classes*.

7. Ameaças à validade

Nesta seção, iremos discutir a validade dos resultados obtidos. Iremos explanar os problemas referentes à validade dos experimentos considerando quatro categorias.

7.1. Validade Interna

Este estudo foi realizado com base no estudo anterior [2]. Os participantes são alunos de pós graduação, matriculados na disciplina de engenharia de software experimental e já haviam lido o estudo original antes de realizar a execução do experimento. O fato de ter conhecimento prévio sobre como seria realizado o experimento e quais eram as classes que foram consideradas *god classes* pelos oráculos no estudo original pode influenciar a resposta dos participantes.

A participação no experimento era voluntária, mas a participação no treinamento sobre as ferramentas que foram utilizadas era obrigatória, resultando na reprovação do aluno caso não participasse. Desta forma, a participação no experimento é vetada caso o aluno não tenha ido para o treinamento com as ferramentas, diminuindo as chances de ocorrer um enviesamento no experimento por conta da falta de conhecimento das ferramentas ou do conceito de *god classes*. Somente um dos alunos realizou o treinamento e não conseguiu realizar o experimento.

Foram utilizadas três ferramentas: A IDE Eclipse, o JDeodorant e o plugin (Decode) desenvolvido pelo pesquisador Waldemar Pires Neto para marcação dos tempos. A ferramenta Eclipse foi utilizada por ser uma IDE gratuita e é a mais utilizada para desenvolvimento Java, com 48.2% de adoção, seguido pelo IntelliJ com 43.6% [7]. Caso o usuário não fosse familiarizado com o Eclipse, o treinamento antes do experimento tinha como um dos objetivos apresentar as funcionalidades necessárias para a realização do experimento. O plugin Decode era responsável por atribuir unidades experimentais (projetos) aos participantes, informar quando o JDeodorant deveria ser utilizado, e contabilizar todo o tempo das atividades executadas pelos participantes. O treinamento realizado antes do experimento utilizando a IDE Eclipse diminui os impactos dessa ameaça.

Como forma de minimizar o impacto de conhecimento prévio sobre os problemas e o aprendizado dos participantes durante a execução do experimento, todos os programas têm uma complexidade baixa, de forma que os participantes conseguissem entender o funcionamento do programa.

7.2. Validade Externa

O número de participantes é estatisticamente pequeno (onze participantes) e todos os participantes são alunos de pós-graduação, o que o diminui significativamente a capacidade de generalização dos resultados encontrados neste estudo. Contudo, os participantes possuem em sua maioria experiência profissional e todos possuem experiência no uso de Java, o que indica uma familiaridade com a linguagem e uma menor probabilidade de ignorar fatores importantes devido ao desconhecimento da linguagem utilizada. Finalmente, embora todos os participantes estivessem matriculados numa mesma disciplina de pós-graduação, eles possuem background/especializações diferentes. Há participantes com foco em estatística, até banco de dados, e não apenas desenvolvimento e engenharia de software, que o contribui para uma maior diversidade da amostra.

Java foi escolhido como linguagem do experimento por se tratar de uma das linguagens orientadas à objeto mais utilizadas. Contudo, isso pode dificultar que generalizamos para todas as linguagens de programação. Trabalhos futuros podem replicar o experimento utilizando outras linguagens.

Apesar de não refletir o software utilizado na prática, os programas são pequenos (<2kLOC), contam com poucas classes e os algoritmos utilizados são simples, foi considerado que se desenvolvedores têm dificuldade em identificar *god* *classes* mesmo em programas simples, intuitivamente, eles também terão em programas mais complexos.

7.3. Validade de Conclusão

Para chegar a conclusão de que não há diferença significativa no esforço utilizado para encontrar *god classes* com relação a utilização do *plugin jDeodorant* foram utilizadas as informações de tempo anotadas no questionário. Infelizmente não foi possível ter respaldo também das medidas de tempo automáticas, devido a uma falha no *plugin* durante a execução do experimento.

Houve cuidado para verificar a normalidade desses dados, feito com o teste de Shapiro-Wilk. Como foi identificado que as distribuições não são normais, foi-se então utilizado o teste não-paramétrico de Wilcoxon para comparação de médias.

Com relação à contabilização e rankeamento dos motivos mais utilizados para identificar as classes como *god class*, foi utilizada uma estratégia de divisão de peso quando diferentes motivos eram utilizados em uma mesma classe. Isso diminui o *bias* para quando vários motivos são escolhidos para uma mesma classe. Apesar disso, ainda haverá a possibilidade de um mesmo motivo ser utilizado inúmeras vezes para classes diferentes, podendo assim haver um enviesamento caso algum participante utilize o mesmo motivo diversas vezes nas classes em que analisou. Contudo, as frequências encontradas no experimento estão de acordo com o do experimento original. Acreditamos assim que neste experimento, por conta do treinamento obrigatório, esse *bias* foi amenizado, não havendo assim um participante que marcasse de forma muito diferente que os demais, enviesando o experimento.

7.4. Validade de Construção

Em relação à validade de construção, observamos o fato dos participantes estarem cientes do objetivo dos experimentos, o que pode ter deixado os participantes mais atentos para identificação das *god classes.* Ressaltamos que mesmo em um experimento para encontrar *god classes*, as respostas não foram homogêneas, o que indica a necessidade de melhor familiarização do conceito por parte dos desenvolvedores para que possam ser evitadas essas práticas.

Todos os experimentos foram realizados com acompanhamento de um supervisor, o que garantia o foco dos participantes na realização do experimento, uma vez que distrações durante a realização dos experimento poderia distorcer o tempo consumido pelos participantes para realizar a tarefa.

O questionário contou com um campo para indicar outros motivos além daqueles listados. Caso o participante identificasse algum motivo que não estava listado, ele poderia descrever o motivo que ele achasse adequado. Apesar do enviesamento inerente da ordem das respostas de um questionário, ao deixar uma alternativa aberta, tentamos minimizar o enviesamento das respostas.

Referências

[1] Fowler, M. (2009). Refactoring: improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] Santos, J. A., de Mendonça, M. G., & Silva, C. V. (2013, April). An exploratory study to investigate the impact of conceptualization in god class detection. In Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (pp. 48-59). ACM.

[3] M. Mantyla. An experiment on subjective evolvability evaluation of object-oriented software: explaining factors and interrater agreement. In ISESE, pages 287–296. IEEE, 2005.

[4] M. V. Mantyla and C. Lassenius. Drivers for software refactoring decisions. In ISESE, ISESE '06, pages 297–306, New York, NY, USA, 2006. ACM.

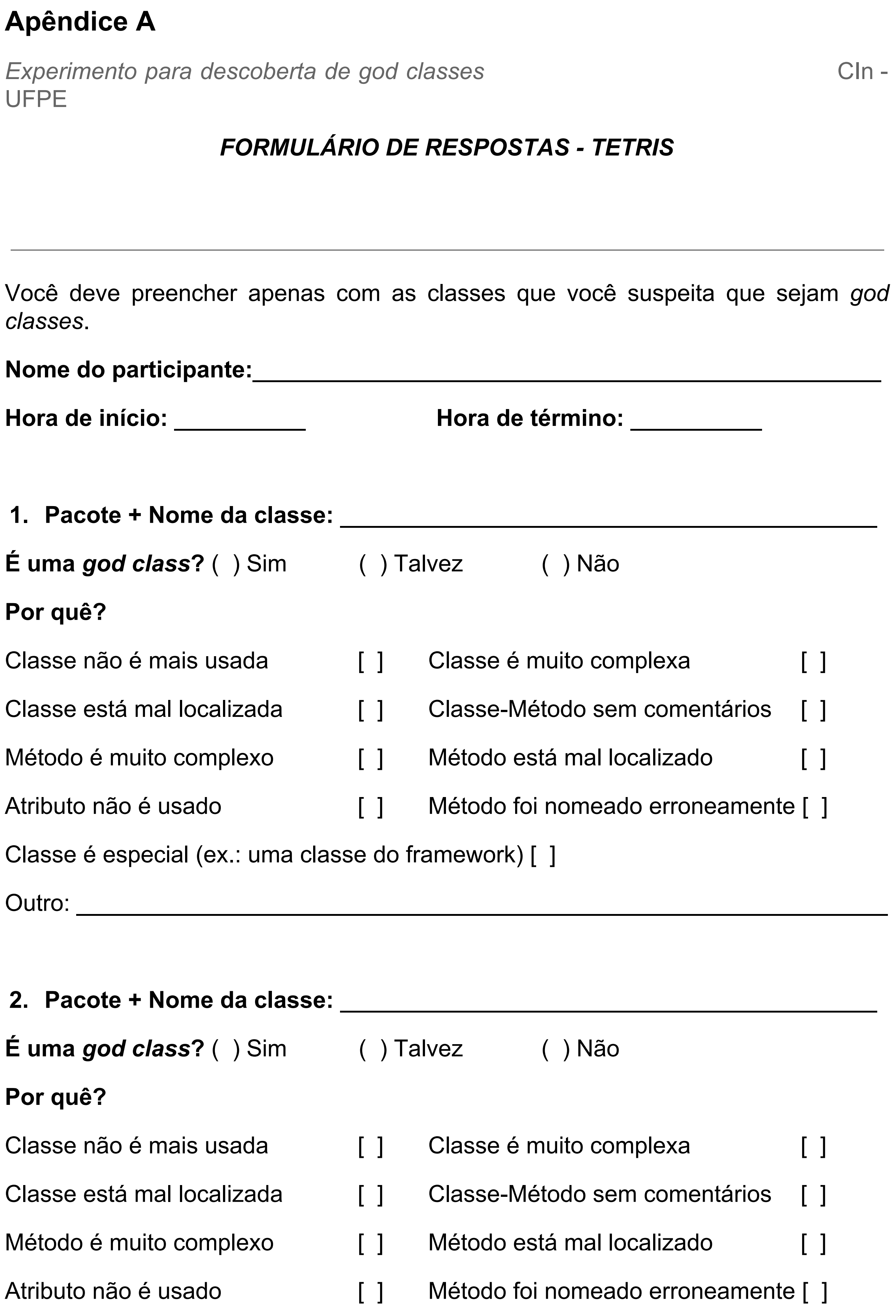
[5] J. Schumacher, N. Zazworka, F. Shull, C. Seaman, and M. Shaw. Building empirical support for automated code smell detection. In ESEM, ESEM '10, pages 8:1–8:10, New York, NY, USA, 2010. ACM.

[6] The Eclipse Foundation. (2016). JDeodorant. Retrieved from https://marketplace.eclipse.org/content/jdeodorant

[7] The Market Share of Java IDEs in Q2 2016. Retrieved from http://www.baeldung.com/java-ides-2016

[8] J. Fleiss et al. Measuring nominal scale agreement among many raters. Psychological Bulletin, 76(5):378{382, 1971.

[9] R. H. Finn. A note on estimating the reliability of categorical data. Educational and Psychological Measurement, 30:71.76, 1970.



Apêndice A

Experimento para descoberta de god classes CIn - UFPE

FORMULÁRIO DE RESPOSTAS - TETRIS

Você deve preencher apenas com as classes que você suspeita que sejam *god classes*.

Nome do participante:____

Hora de início: __________ **Hora de término:** __________

1. **Pacote + Nome da classe:** ____

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

2. **Pacote + Nome da classe:** ____

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

3. Pacote + Nome da classe: ______________________________

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

4. Pacote + Nome da classe: ______________________________

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

5. Pacote + Nome da classe: ______________________________

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]

Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

6. Pacote + Nome da classe: ______________________________

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

7. Pacote + Nome da classe: ______________________________

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

8. Pacote + Nome da classe: ______________________________

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

9. Pacote + Nome da classe: ______________________________________

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____

10. Pacote + Nome da classe: ______________________________________

É uma *god class*? ( ) Sim ( ) Talvez ( ) Não

Por quê?

Classe não é mais usada	[ ]	Classe é muito complexa	[ ]
Classe está mal localizada	[ ]	Classe-Método sem comentários	[ ]
Método é muito complexo	[ ]	Método está mal localizado	[ ]
Atributo não é usado	[ ]	Método foi nomeado erroneamente	[ ]

Classe é especial (ex.: uma classe do framework) [ ]

Outro: ____