Universidade Federal da Bahia
Instituto de Matemática

Programa de Pós-Graduação em Ciência da Computação

BASIC NOTIONS OF REPLICATION CHALLENGES: A VIEW ON MULTIPLE REPLICATIONS OF A HIGHLY-CONFIGURABLE SYSTEMS EXPERIMENT

Daniel Amador dos Santos

DISSERTAÇÃO DE MESTRADO

Salvador
September 2, 2020

DANIEL AMADOR DOS SANTOS

BASIC NOTIONS OF REPLICATION CHALLENGES: A VIEW ON MULTIPLE REPLICATIONS OF A HIGHLY-CONFIGURABLE SYSTEMS EXPERIMENT

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Advisor: Eduardo Santana de Almeida

Salvador
September 2, 2020

Sistema de Bibliotecas - UFBA

Amador dos Santos, Daniel.

Basic Notions of Replication Challenges: A view on Multiple Replications of a Highly-Configurable Systems Experiment / Daniel Amador dos Santos – Salvador, 2020.

75p.: il.

Advisor: Prof. Dr. Eduardo Santana de Almeida.

Dissertação (Mestrado) – Universidade Federal da Bahia, Instituto de Matemática, 2020.

1. Replication. 2. Highly-Configurable Systems. 3. Constant Comparison, 4. Empirical Software Engineering. I. Santana de Almeida, Eduardo. II. Universidade Federal da Bahia. Instituto de Matemática. III Título.

CDD – XXX.XX

CDU – XXX.XX.XXX

TERMO DE APROVAÇÃO

DANIEL AMADOR DOS SANTOS

BASIC NOTIONS OF REPLICATION CHALLENGES: A VIEW ON MULTIPLE REPLICATIONS OF A HIGHLY-CONFIGURABLE SYSTEMS EXPERIMENT

Esta Dissertação de Mestrado foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia.

Salvador, 02 de Setembro de 2020

Prof. Dr. Eduardo Santana de Almeida
Universidade Federal da Bahia

Prof. Dr. Manoel Gomes de Mendonça Neto
Universidade Federal da Bahia

Prof. Dr. Rafael Prikladnicki
Pontifícia Universidade Católica do Rio Grande do Sul

ACKNOWLEDGEMENTS

There is a proverb saying that “a journey of a thousand miles begins with a single step”. I would rather say that the journey begins way before than any step is given. I would like to thank God for bringing me to the starting point of this road and making me living unmatching experiences throughout the path. I am sure those experiences will be useful for the future challenges coming and now are a part of the person I became.

Along the road, I met a wonderful person, who has been immensely special since then. Thank you, my dear Brenda Martins, who has been so patient and understanding, for always motivating me.

I would like to thank my advisor, Professor Eduardo Almeida, who had given the vision for this research and accepting me as a master’s student. Thanks for being my guide on this journey.

I thank my loyal friends who have been with me all the way: Denivan Campos, Beatriz Rêgo, Filipe Adeodato, Edilton Santos, and Lucineia Batista. All the help, support, and remarkable moments we have shared together made all the difference. Surely, I owe part of my success to you.

Thanks to my family for believing this dream could be true a long time ago before I came to UFBA and Salvador. Thanks for all the support, my mother Salma Rodrigues, my father Dinamar Amador Junior, my brother Dinamar Amador Neto (who provided a brand new laptop for working on this dissertation).

I could not forget to express my gratitude to the members of the RiSE group and the participants of the empirical study. You were quite sympathetic in participating in such a long replication.

This story is finally coming to an end. Others will come. Once again, I thank all the people aforementioned and a thousand more which I would not be able to list here. May the everlasting God bless you all the way he blessed me.

If you want to change the world, you must be your very best in the darkest moment.

—ADMIRAL WILLIAM H. MCRAVEN

RESUMO

À medida em que a Engenharia de Software Experimental cresce em maturidade e número de publicações, mais replicações são necessárias para fornecer um fundamento sólido para as evidências encontradas em pesquisas anteriores. No entanto, em geral, estudos de replicação são escassos, e alguns tópicos sofrem mais que outros. Além disso, os desafios associados à replicação de estudos experimentais não são bem compreendidos. Neste estudo, pretende-se preencher esta lacuna investigando as dificuldades que surgem ao se replicar um experimento. Faz-se uso de uma metodologia inovadora, no qual participantes com experiências distintas desempenham o papel de um grupo de pesquisa tentando replicar um estudo experimental. No total, foram feitas oito replicações. É utilizado o método da comparação constante de teoria fundamentada para análise qualitativa. Observou-se nas replicações que a maioria dos resultados se sustenta ao se comparar com os experimentos originais. Entretanto, os participantes relataram diversas dificuldades, a maioria delas relacionadas à clareza das instruções e à qualidade dos artefatos de replicação. Baseado em nossa experiência, também são fornecidas recomendações, que podem ajudar a mitigar problemas relacionados à replicação de estudos experimentais.

Palavras-chave: Replicação, Sistemas Altamente Configuráveis, Comparação Constante, Engenharia de Software Experimental

ABSTRACT

As Empirical Software Engineering grows in maturity and number of publications, more replications are needed to provide a solid grounding to the evidence found through prior research. However, replication studies are scarce in general and some topics suffer more than others with such scarcity. On top, the challenges associated with replicating empirical studies are not well understood. In this study, we aim to fill this gap by investigating difficulties emerging when replicating an experiment. We used an innovative method in which participants with distinct backgrounds play the role of a research group attempting to replicate an experimental study. Eight replications in total were performed. We used the grounded theory's constant comparison method for qualitative analysis. We have seen in our replications that most results hold comparing with the original experiments. However, the participants reported many difficulties, mostly related to the clarity of the instructions and the quality of the replication artifacts. Based on our experience, we also provide recommendations that can help mitigating issues related to experiment replication.

Keywords: Replication, Highly-Configurable Systems, Constant Comparison, Empirical Software Engineering

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ESE	Empirical Software Engineering
HCS	Highly-configurable System
SPL	Software Product Line
RQ	Research Question
VM	Virtual Machine
UFBA	Federal University of Bahia

Chapter 1

INTRODUCTION

1.1 MOTIVATION

In empirical sciences, it is not enough that a phenomenon be only reasonable and logically consistent, it is also important to have evidence for that. Moreover, for a knowledge to be considered valid, it is necessary that a phenomenon can be replicated and observed. (JURISTO; MORENO, 2001).

Over the years, many guidelines, frameworks, and techniques have been developed to guide researchers to perform replications in Software Engineering (MAGALHAES et al., 2015). Furthermore, to a limited extent, experiment papers have been published providing their assets for replication in the form of a replication package (SOLARI; VEGAS, 2006). However, many questions remain unanswered, such as: If a researcher selects an experiment paper to perform a replication, does that mean they would be able to execute that replication seamlessly? If not, what problems and difficulties that researcher would face when conducting the replication? Would it be possible to obtain the same results the authors from the original paper obtained? Those questions have motivated our study.

In general, replication studies are scarce in Software Engineering, and some sub-areas suffer more than others. Highly-configurable Systems (HCSs), an established and popular area in Software Engineering, is one such area suffering from a lack of replication studies even with recent efforts to conduct more empirical studies in the area (BAGHERI et al., 2016; SPLC, 2019). We know some of the challenges related to replicating studies of non-HCS experiments (MENDE, 2010), however, HCSs are perceived to be more complex compared to non-HCS (BASTOS et al., 2017) and challenges associated with replicating an HCSs study are yet to be investigated. Is replicating HCSs studies more difficult? Which unique challenges does that impose on the researchers? Do scientists with different level of research experience face different replication difficulties? These are some of the questions that are also unanswered.

1.2 OBJECTIVE

In summary, this research aims to investigate the possible problems that may happen in an Software Engineering replication and more specifically, in an HCS experiment replication.

Additionally, this research intends to investigate the original experiment on the following aspects: if the experiment replication is viable, and if the replication outcome matches the baseline results. Thus, while we designed a group of replications to observe possible replication difficulties, we were also interested in the results of the experiment itself.

1.3 RESEARCH METHOD

To fulfill the mentioned gaps and answer the questions, we conducted a multi-method study. This research consists of a set of experiment replications, focus group sessions with the participants, and constant comparison analysis over the data collected. The study was conducted with eight groups, with each group performing a replication (i.e., eight replications in total). The groups were formed by three participants playing the role of researchers. We selected participants having at least a minimum background on Software Engineering research and/or HCS. However, we purposely gathered teams with distinct levels of research expertise in order to capture different perceptions about replication difficulties. After a paper selection process involving inclusion criteria (listed in Subsection 4.1.1), we chose an experiment that investigated sampling algorithms' performance for detecting variability bugs. (MEDEIROS et al., 2016)

After replicating the experiment, we conducted a focus group session with every team, so that we could gather information regarding the difficulties they experienced during the replication.

Based on the focus groups transcripts and other artifacts obtained from the experiment replications, we conducted a qualitative analysis using the constant comparison method (STRAUSS; CORBIN, 1998). We then defined a set of phenomena and explanations for replication problems collected from the participants, as well as other phenomena involving their experience about the study. From those explanations, a set of recommendations was extracted for practitioners, researchers and educators.

1.4 STATEMENT OF THE CONTRIBUTIONS

This dissertation makes the following contributions:

1. Introduce an empirical study where we bring together experiment replications, applied focus group sessions to assess the difficulties the participants experienced, and used the constant comparison method to analyze the qualitative data obtained from the focus group sessions.

2. Perform eight replications of the study *A Comparison of 10 Sampling Algorithms for Configurable Systems* (MEDEIROS et al., 2016). We aimed at verifying if the original study results hold under a group of external replications.

3. Document challenges and difficulties when replicating an experiment in Software Engineering, taking into account different levels of expertise of the participants. Based on the evidence found, we provide recommendations for researchers, educators, and practitioners.

1.5 STRUCTURE

The remainder of this dissertation is structured as follows:

Chapter 2 discusses the background of this dissertation, which provides foundation for comprehending the subjects addressed in this work.

Chapter 3 shows the empirical study's design.

Chapter 4 describes the original experiment, from the selection process to the explanation of its design and results.

Chapter 5 describes the operations of the empirical study.

Chapter 6 describes the empirical study's results, recommendations, and threats to validity. This chapter also presents an additional ad hoc investigation we performed about teaching practices in Empirical Software Engineering.

Chapter 7 presents the conclusions of the empirical study and directions for future research.

Chapter 2

BACKGROUND

In this chapter, we lay the foundation of our work: we give an introduction about Empirical Software Engineering (ESE) (Section 2.1), in which we explain quantitative and qualitative research (Subsection 2.1.1), underline the definitions of replication (Subsection 2.1.2) and their types (Subsection 2.1.3), and present the current state of replication in Software Engineering research (Subsection 2.1.4). We also introduce the theme of Highly-configurable System (HCS) (Section 2.2), present an example to illustrate its concepts (Subsection 2.2.1), and we address testing on HCSs (Subsection 2.2.2).

2.1 EMPIRICAL SOFTWARE ENGINEERING

Software is an integral part of our world. Practically almost every human activity is aided by computing nowadays. From the most basic to the critical ones, there is constant need for reliable, safe, and user-friendly software. Software Engineering is defined as "the engineering discipline that is concerned with all aspects of software production" (SOMMERVILLE, 2011).

Since software holds such importance in the world economy, it would be reasonable thinking that Software Engineering relies heavily on science to deliver the best products and practice. Unfortunately, not that much. Although industry and research offer a multitude of methods, tools, and technologies, there is little evidence to support the decision of practitioners who need to choose a particular one (JURISTO; MORENO, 2001).

Software Engineering intersects with various other computer science areas. Therefore, it is a multidisciplinary subject. However, Software Engineering is relatively young if compared to other engineering disciplines. Several practices used in the current state have no grounding on scientific methods (JURISTO; MORENO, 2001). So, many of the widely dominant methodologies and tools employed have their efficiency attested only by empiricism. It is needed to generate a body of knowledge about the phenomena that are unique to Software Engineering.

Usually, the experimental methods show themselves fruitful for investigating people and their work. It does not mean no other method can be used when inquiring about Software Engineering, but social and psychological aspects might get in the way. Moreover, for those themes, empirical methods have a large history of usage and knowledge-yielding (WOHLIN et al., 2012). The application of empirical methods on the research of Software Engineering is known as Empirical Software Engineering.

2.1.1 Quantitative and Qualitative Research

The reality surrounding us can be observed in different manners. As science intends to understand the phenomena surrounding a certain aspect of the study, there may be different paths to get into the knowledge. Let us say, for instance, for physical phenomena, it might be interesting to establish numeric relations and formula to describe the several events that can be observed; a quantitative way to perceive the phenomena. On the other hand. For other kinds of knowledge, such as psychology, it might be interesting to rely on interpretation and explanation of the subjects; a qualitative way to comprehend the phenomena.

Empirical sciences had split both strategies to acquire knowledge in quantitative and qualitative research methods. Quantitative research focuses on drawing cause and conclusions between variables involved in a study and their outcome. Qualitative research relies on people perceptions (which in this case, people are the participants of a research) about the phenomena, and their description of how the events happen. (WOHLIN et al., 2012)

In many occasions, Software Engineering is highly influenced by human behavior. As Wohlin et al. (2012) state, "we cannot expect to find any formal rules or laws in software engineering except perhaps when focusing on specific technical aspects". Therefore, qualitative data is fundamental for understanding Software Engineering in an empirical way. In other situations, technical aspects makes quantitative data to be reasonable for investigate a certain phenomenon. For instance, a research might intend to assess a certain algorithm's performance. Another example would be assessing metrics from an agile-development team.

On top of that, we can observe that, due the multidisciplinary nature of Software Engineering, research can work with two or more research methods. The combined use of different methods for data collection and analysis is called multi-method or mixed-method. About this approach, Seaman (1999) defends that "in software engineering, the blend of technical and human behavioral aspects lends itself to combining qualitative and quantitative methods, in order to take advantage of the strengths of both". In this direction, Easterbrook et al. (2008), synthesizing Creswell (2002), declare that "all methods have limitations, and the weaknesses of one method can be compensated by the strengths of other methods".

2.1.2 Replications

La Sorte defines that "replication refers to a conscious and systematic repeat of the original study" (SORTE, 1972). According to Juristo and Vegas, replication is "the repetition

of an experiment to double-check its results" (JURISTO; VEGAS, 2009). Although La Sorte's definition is wider in terms of not restricting replications to experiments only, in both definitions there is a mention of repeating a previous study. Most of the definitions about replication agree that it is mandatory to repeat a previous study (MAGALHAES et al., 2015), also called original or baseline study. That is why these authors do not consider approaches that attempt to validate a past study under a completely different methodology, like happens in conceptual replications (BALDASSARRE et al., 2014) and independent replications (KREIN; KNUTSON, 2010).

Hardly ever, findings from a single study can be generalized to all possible contexts. Subjects and the environment might impact the results even when the methodology is strictly designed and followed. In other cases, there might be bias of the research group even when not intended. Therefore, empirical studies should be replicated in order to make evidence valid outside the context of the original study.

Sometimes, a replication might refute an existing study and the authors of the original study might disagree with the refutation, refining their arguments and reaffirming the findings of their study (BERGER et al., 2019). By having another team analyzing the results of a research, there is a double-check on the methodology and findings of an empirical study, which is beneficial for science, in general.

In order to be able to replicate studies in an external context, it is necessary to transmit knowledge from the original group to the replicating one. A useful tool for that are the laboratory packages. Laboratory packages or replication packages are the packed set of information and materials to use in a replication (SOLARI; VEGAS; JURISTO, 2018). Ideally, they should contain every artifact used in the original empirical study so replicating researchers could recreate the original setting as much as possible. Tips and explanations about the original studies procedures are also expected to be found in a replication package. Thus, tacit knowledge transfer problems can be minimized (SHULL et al., 2002).

2.1.3 Replication types

In the Software Engineering literature there are several classifications mentioned based on involvement of the original researchers with the replication and how much a study design is changed. Magalhaes et al. (2015) point out that their naming is not consistent throughout, i.e., sometimes a same term has different meaning depending on the author, while in others different names refer to a same idea.

In terms of involvement of the original researchers, we will use the definitions from Brooks et al. (1996). They classify replications as internal (when the replication is performed at least by one of the researchers from the original study) and external (when none of the researchers leading the replication were present at the original instance).

A study can be replicated as similar as possible to the baseline or can have design changes. The former case is useful to validate findings under different circumstances while the latter attempts to extend the results. Both ways are valid but they have different roles depending on the goals of researchers replicating the study (JURISTO; VEGAS, 2011). Many authors express that it is difficult to make exact replicas in Software Engineering

the way it is made in other science fields such as physics, chemistry, and biology. Human factors have a strong influence on Software Engineering studies. Thus, it is rarely possible to perfectly recreate the same settings of the original study.

Concerning the similarity of baseline and replications, hereafter we will be using classifications from Baldassarre et al. (2014). In their classification, a close replication is the one which is as similar as the original study, while a differentiated replication is the one which have intentional significant design modifications.

There might emerge the belief that only replication papers which confirm the baseline results are valid. On the other hand, there is a movement in the research community to stimulate the performing and publishing of studies that refute the original ones (BERGER et al., 2019). Those papers are useful in a way they show the findings of some study are not valid or have some effects due to unmanaged validity threats. They may indicate also a research tendency the community should not follow or steps to avoid in a certain experiment. Basili, Shull and Lanubile (1999) state that if the community is willing to build a complete and unbiased body of knowledge, it must publish non-confirmatory studies.

2.1.4 Overview of Replications in Software Engineering

In the last years, Silva et al. (2014), Magalhaes et al. (2015), and Bezerra et al. (2015) have been performing tertiary studies about replication in Software Engineering, which are still the most recent with regards to the topic of this dissertation. These tertiary studies were in general aiming to understand which topics are most addressed in replications. They also have cataloged papers containing definitions and frameworks about replication. Thus, those studies can give a view of which research direction the community is investigating and how researchers perceive concepts, techniques and tools concerning the subject.

In these mappings, the following gaps were identified to improve replications, both in number and quality:

1. The number of replications must grow;

2. Researchers must achieve a common ground on classifications and definitions;

3. Over 60% of the replications studies do not cite any papers with guidance about replication. If the community is willing to produce more high quality replications, it should pay more attention to papers about replications.

Although the number of replications still must grow, an increase of publications of this kind is already in course (MAGALHAES et al., 2015). The evolution in replications until 2012 can be seen in Figure 2.1. The creation of the RESER (Workshop on Replication in Empirical Software Engineering Research) also helped in this improvement. Unfortunately, this workshop's last happened in the year of 2013 and it is not active anymore. While there is a growing recognition on the Software Engineering community that providing good quality artifacts (manifest for instance in ICSE's ROSE initiative to award

the best conference's artifacts[1]), we are not aware if this resonates directly in increasing the amount of replications the way RESER did.

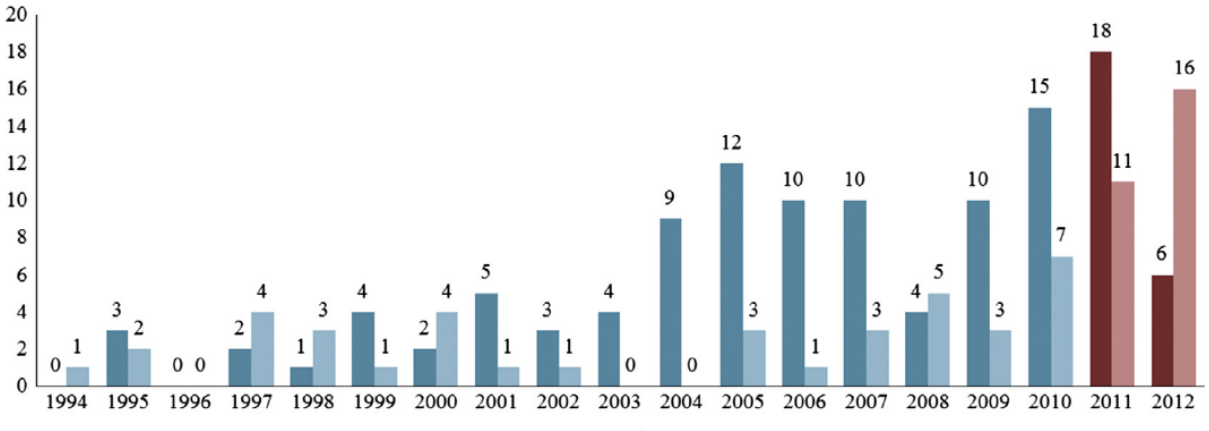

Figure 2.1 Evolution of the number replication studies. SOURCE: (SILVA et al., 2014)

Another study highlights the importance of performing group replications in Software Engineering (SANTOS et al., 2019). This area usually suffers from small samples on their empirical studies and restricted results generalization. Group replications are an effective way to obtain greater sample sizes. The authors of the mentioned paper claim that, while more mature empirical sciences, such as medicine and pharmacology, has sophisticated mechanisms to aggregate evidence from multiple replications, Software Engineering is still scarce of such efficient methods. Thus, we can state that using aggregation methods well suited for Software Engineering (such as the one the mentioned study proposes) is additionally a challenge to overcome.

2.2 HIGHLY-CONFIGURABLE SYSTEMS

A computer system can be designed to adapt itself to different circumstances. Those adaptations can happen on several levels, ranging from modeling level to implementation details in the source code (Berger et al., 2013). The same software may be tailored to run in different platforms, operating systems, and even to suit different user needs (SPENCER; COLLYER, 1992). We refer to those systems as HCSs. Han and Yu (2016) states that HCSs allows "users to customize a large number of configuration options while retaining a core set of functionality".

In a similar direction, Software Product Lines (SPLs) is a related approach (ABAL; BRABRAND; WASOWSKI, 2014), referring also as a set of systems sharing a common core offering variations to specialize software for different needs (POHL; BÖCKLE; LINDEN, 2005). SPL has been a relevant topic in the research community, with an extensive set of definitions and methodologies used for understanding and developing systems with configuration in focus. Therefore, in this dissertation we are assuming SPLs and HCSs

[1] ⟨https://2020.icse-conferences.org/track/icse-2020-rose⟩

can be treated as correspondent, relying on Sincero et al. (2007) affirmation that HCSs are able to achieve "the same technical goals that the SPL guidelines aim at."

In HCSs and SPLs, the software construction is conceptualized around features. A feature can be described as "a unit of functionality of a software system that satisfies a requirement" (APEL; KÄSTNER, 2009). Thus, each unit of functionality (a.k.a feature) will be identified to develop the software system. A set of features will be present in every configuration of the software (mandatory features), and other features will be present only in specific cases (optional features). Each combination of features forming a working software system is called a configuration.

In addition to the HCSs, Software Product Lines area has conceived a development methodology, known as Software Product Line Engineering (SPLE) (POHL; BÖCKLE; LINDEN, 2005). From requirements elicitation to testing and delivery, all the software development is conceived around generating products from the same set of artifacts.

2.2.1 Example

We present here an example of a Software Product Line to depict the concepts presented so far and introduce new ones. The example's feature model can be seen in Figure 2.2. The diagram presents a family of software that can sell tickets for bus rides, flights, and boat trips. The mandatory features are indicated by rectangles solid circles, and the optional features are indicated by rectangles with empty circles. In the lower part of the Figure, further relations between features are listed using the following notation: an arrow indicates dependency and a caret means "AND". Thus, for instance, the first line could be read as "*Plane depends* on *Multiple Segments* AND *Stop Over* AND *Connection*".

Besides mandatory and optional features, there is the concept of alternative features. They can be defined as a set of features that only one of the set can be present in a given configuration. In the model presented, the family will be able to generate a product selling tickets for plane, boat, or bus (represented by the homonym features), but only one transportation method will be available at a time.

Some features can be enabled only if other specific ones are present. These are called dependent features. In the example, *Luggage Registration* can be enabled only if *Register Passenger* is present. On the other hand, Plane feature depends on *Multiple Segments*, *Stop Over*, and *Connection*. Dependent features do not necessarily need to be children of the same parent. However, a child feature necessarily is dependent of its parent node.

2.2.2 Testing in Highly-Configurable Systems

In Quality Assurance, Software bugs are considered faults and defects existing in software. A fault is an incorrect instruction in the computer program. It does not necessarily manifest itself in program execution. When it does manifest, then that defect caused an error, which is an incorrect program state. One of the challenges of quality assurance in software is that a fault can be silent for a long time, but can induce an error in a very specific circumstance, making it then difficult to reproduce the bug.

The referred situation can often happen in an HCS. Defects can be introduced, but might appear as an error only in specific configurations. This is an example of

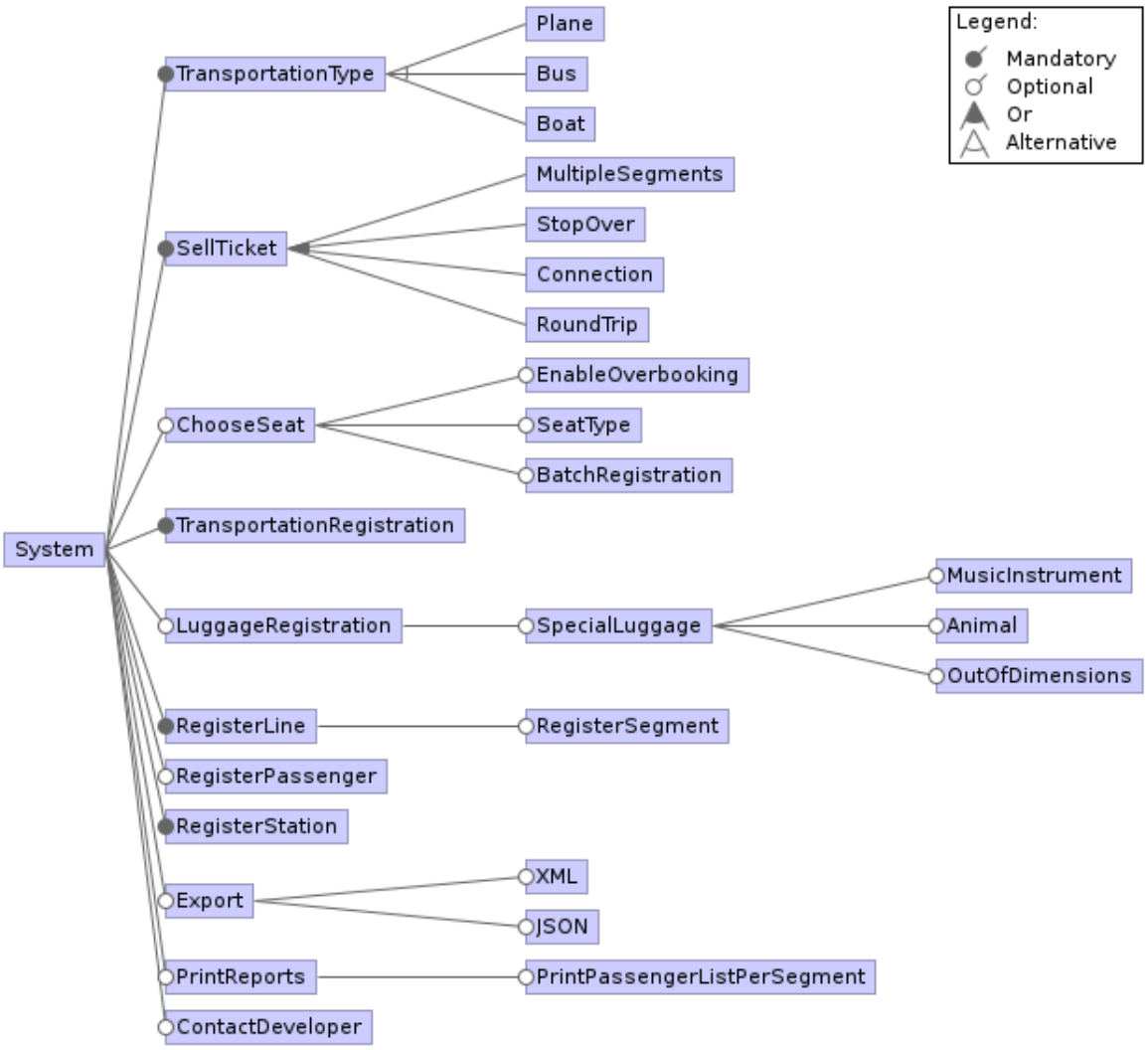

Figure 2.2 Example of a feature model

variability bug. The ideal solution for preventing those bugs would be testing all the features combinations possible. However, depending on how much optional features a configurable system offers, it might infeasible to test all the possible products generated, even with automated approaches. Since the early 1990s, it is known that it might be difficult to test all the configuration combinations in a configurable system (SPENCER; COLLYER, 1992). For instance, a statement in this research field mention the Linux kernel, a large HCS, can yield more combinations than atoms existing in the universe (MEDEIROS et al., 2016).

The research community has been actively producing research addressing development and testing in HCS to bring answers to the challenges that this kind of software imposes. Since testing all configuration combinations in many cases is infeasible, one strategy is using sampling algorithms for testing (PERROUIN et al., 2010). Those algorithms select subsets of configurations that are representative of the complete set. The different algorithms can aim different strategies, sometimes aiming for better computation time and others for the size of the samples (increasing the test coverage).

2.3 CHAPTER SUMMARY

We presented in this chapter the foundation themes of this dissertation. We introduced replication definitions and a brief overview of replication research in Empirical Software Engineering. We also addressed HCSs, which is an object of study of the replications proposed. The next chapter presents the design of our empirical study proposed.

Chapter 3

EMPIRICAL STUDY DESIGN

In this chapter, we describe the design of our empirical study. We start from the research questions elicitation (Section 3.1). After that, we explain the experiment replication phase (Section 3.2), the focus group session strategy to gather feedback from the participants (Section 3.3), and list the artifacts used in this empirical study (Section 3.4). At last, we explain the data analysis planned for this study (Section 3.5), showcasing the usage of the constant comparison method for qualitative feedback obtained primarily from the focus group sessions.

3.1 RESEARCH QUESTIONS

The main objective of this work is to investigate possible replication difficulties of a Software Engineering experiment. While it is important to verify if the results of the baseline study confirms under a different context, we are also interested in identifying which experiment phase presents the most challenges, if there are problems specific to HCSs, and so on. In this study, we answer the following research questions:

RQ1 Are there problems or difficulties experienced in a Software Engineering experiment replication?

RQ1.1 In case there are problems, which experiment phases present more difficulties?

RQ2 Are there problems specific to a Highly-configurable Systems experiment replication?

RQ3 Does the experiment confirm the baseline study results?

In order to maximize the feedback from researchers attempting to replicate an experiment, we designed an empirical study where several participants replicate a study, essentially playing the role of researchers. We assign them a task to replicate the original

study, and then we collect their feedback at every step of the experiment. The participants are split into groups, each one independent from the other, thus every team is emulating a team of researchers. Since we have several independent groups obtaining their own replication results, we consider that independent replications are performed with this design.

We selected the paper "A Comparison of 10 Sampling Algorithms for Configurable Systems" (MEDEIROS et al., 2016) as our final choice for the experiment replication (Chapter 4). This study has five authors. Since in our observatory study the participants are not expected to report the findings themselves to any kind of scientific publication and the design is already provided to them, we divide the participants into groups of three. Figure 3.1 shows the high level overview of inputs and outputs of our study.

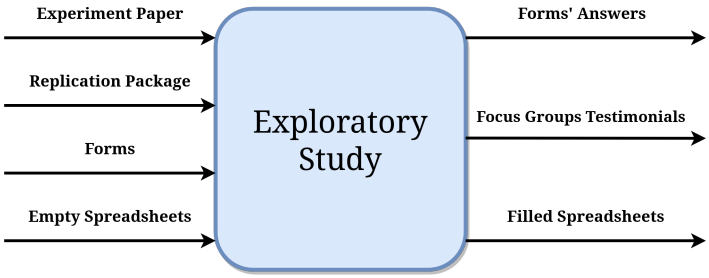

Figure 3.1 Empirical study input/output

3.2 EXPERIMENT REPLICATION PHASE

Before designing how the replication activity would be set up, we tested the replication package provided on the paper's website. It was performed in order to guarantee that the experiment is doable from the beginning until the end (even if problems occur, but none of them preventing the replication to happen).

Each team is given the task of replicating the experiments described by Medeiros et al. (2016) The participants are expected to complete the task of executing the algorithms on the subject systems, as described in the original paper. Therefore, they have to use the assets provided with the replication package. In order to collect the results they obtain, a spreadsheet is given to them. The estimated time for them to execute the algorithm and collect the results is one week (counting only the execution part).

3.3 FOCUS GROUP

After the participants finish the study, we perform a focus group session with each team. The goal is to extend the feedback provided by them from the forms. The sessions are designed to be approximately 45-minute long. The sessions have audio (two sources) and video recordings, according to the participants' consent.

The approach of the sessions was planned to be somewhere between a conversation and a non-structured interview. We have a set of questions asked to the participants (which

we provide in Appendix A). However, the participants are allowed to interact with the other ones, in a way they can complement each other's answers or disagree about some comment said by the others. The order of the questions can be changed during the focus group session, if for example, a subject pertaining to a subsequent question is mentioned beforehand. After all sessions are complete, we transcribe the focus group instances to text.

3.3.1 Session Structure

The focus group session is divided in four sections, each one containing questions about the activities conducted by the participants: environment set up, algorithms execution, paper interpretation and questions about the participant's view on the replication experience.

Environment set up refers to the phase in which the participants install the appropriate software necessary for running the replication. This involves installing the virtual machine and Operating System (in case participants wish to do so), installing the libraries and Integrated Development Environment (IDE). This phase also involves running one sampling algorithm so that participants can assure all dependencies were correctly installed.

The algorithms execution phase contains questions about the core experiment execution, once the environment is properly set up. This also involves how the participants perceive instructions related to running the sampling algorithms.

As the name suggests, paper interpretation questions refer to gauging participant's perception about understanding the paper. Among other things, this involves their perception about the language of the paper, if it is easy to understand and if there are some parts that were not clear.

Finally, we introduce questions about participants overall impressions about participating in the experimental study. Our intention is to perceive how motivated participants are and general feedback about the whole study itself. The intention behind this session is to enrich the other sessions results, whenever possible.

3.4 ARTIFACTS

In this section, the artifacts of this exploratory study are described. All assets are listed on Table 3.1.[1].

The exploratory study has three forms: a participant characterization form, a form gathering feedback from the environment set up phase, and another one extracting feedback from the algorithms execution.

The participant characterization form is used to investigate the participants' backgrounds and skills. The teams are assembled based on participants profiles. Our goal is that teams have at least one member familiar with programming.

The form about setting up the environment intends to extract possible difficulties

[1]The assets produced in this study are available in ⟨https://github.com/danielamador/ese_replication_assets⟩

Table 3.1 Artifacts Provided to the participants

Artifacts
participant characterization form (Form 1)
Environment setting up form (Form 2)
Algorithms execution form (Form 3)
Experiment data extraction spreadsheets
Virtual Machines with Ubuntu 14.04
Paper and link with replication assets

experienced in that phase. Similarly, the form about algorithms execution intends to capture the adversities faced.

A spreadsheet is given to the participants. Thus, they can run the algorithms in the replication package and fill the cells with the execution results for each algorithm. This asset is useful to verify whether the participants achieve the same results as the baseline study or the outputs are divergent.

Virtual Machines (VMs) with default Ubuntu 14.04 installations can provided. The reason for that is, due to the short available time to perform the replication, the participants might get stuck, since the dependencies work preferably on this Linux distribution. We assumed that figuring out how to install the dependencies on Windows or Mac can cost the participants too much time. However, using the VMs is not mandatory, but a commodity provided to participants who do not desire to install Ubuntu 14.04 directly on their computers.

3.5 DATA ANALYSIS

In order to enrich and look for explanations about the data obtained in the replication, we use qualitative data analysis techniques to synthesize evidence primarily from focus group transcriptions. Additionally, we use the other artifacts to triangulate data obtained from the focus group sessions: the spreadsheets and the forms answered by the participants.

We might be watchful to detect patterns present on the groups feedback and also divergences in the groups answers. Thus, our intention with that is to form a solid explanation of the difficulties faced by participants when replicating the proposed experiment.

3.5.1 Constant Comparison Method

We apply the constant comparison method for analysis. We believe this method is suitable because we primarily rely on qualitative data: the transcripts from the focus group sessions. We have other artifacts, which were also taken into consideration when applying this grounded theory technique, like feedback forms applied after the experiment and explanatory notes written by the participants on the extraction spreadsheets.

The process, as described by Strauss and Corbin (1998), is composed of open coding, axial coding and selective coding. Although the authors argue that those phases are not necessarily sequential, we are describing them as if they were, for simplification purposes.

As Strauss and Corbin (1998) proceeds on their explanation, they define coding as "the analytic process through which data are fractured, conceptualized and integrated to form theory". They also state that each code represents a phenomenon.

The open coding is performed per line. Each line might be describing phenomena besides the subject of the question. The line, then is labeled according to the participant it is talking about. A line can receive none, one or more labels, depending on how many topics the participant addresses in a single sentence. The labels generated must be always in the positive form. So, if for example a participant complains the paper is not written in a clear language, the code can be named as "clarity on the paper" instead of "absence of clarity on the paper". This is aligned with Seaman's point of view (SEAMAN, 1999), that states: "none of the codes relates a value, just a concept."

Sometimes, the participants might mention a subject of some question asked in a different moment. For example, it might be mentioned a difficulty of setting the environment during a question concerning algorithms execution. Addressing that, each focus group session is expected to be coded twice: one for verifying the emerging codes and other for tagging those "displaced" answers.

There can be no line without a code. For the cases some line or group of lines is not related to any code, there is the "no code" tag. That is performed to make text search easier on the analysis tool, so we can differentiate uncoded responses from text that is not useful to the data analysis. Unintelligible answers and questions asking for clarifications about some misheard words are not taken into consideration; therefore, they are ignored. Random chatting about themes not related to the experiment, computer science or scientific research are ignored as well.

Axial code refers to ensemble the excerpts extracted from the data sources and rearranging them in order to understand phenomena and their variations. It consists of establishing logical links between the codes created in the open code phase so one can understand the variation of each phenomena and how categories are related.

Strauss and Corbin (1998) have developed a tool for performing axial coding called "paradigm". This tool helps the analyst to visualize the conditions of a phenomenon, the actions/interactions in which the participants use in response to that phenomenon, and the outcomes of those actions/interactions, called consequences. In this work, we use a simplified form of the paradigm, which comprises the classifications mentioned above and others more as showed in Figure 3.2.

During axial coding, we start by arranging the coded sentences from the open coding step according to the phases of the paradigm, indicating the cause of a phenomenon, a phenomenon itself, action/interaction strategies the participant took when facing the phenomenon, and the consequences of the phenomenon. Next, we synthesize explanations from each coded sentence and established the links between the codes.

It is important to highlight that this dissertation did not aim to generate a full-fledged theory about difficulties in replicating experiments. So, in this study there are no iterative rounds of focus group sessions, refining the codes until reaching a saturation point as the grounded theory literature advocates. The goal of using constant comparison method is to provide an initial glimpse of difficulties in replicating a Software Engineering experiment and to provide explanations about divergences on the quantitative results obtained in

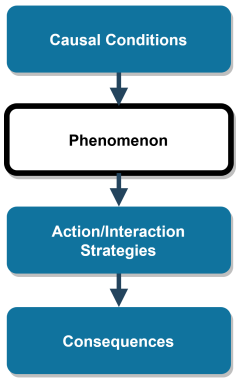

Figure 3.2 Simplified version of the paradigm

case they exist.

3.6 CHAPTER SUMMARY

This chapter presented our empirical study's design. This design comprises the following elements: research questions, experiment replication, focus group sessions, and the use of constant comparison for qualitative data analysis. We have listed the artifacts needed to operate the study as well. The next chapter presents the original experiment's selection process, design, and results.

Chapter

4

ORIGINAL STUDY

In this chapter, we describe the original study used for the replications. The following sections present the process for the paper's selection (Section 4.1), its experimental design comprising the study's goal and research questions (Section 4.2), and its results (Section 4.3).

4.1 REPLICATION SELECTION PROCESS

In this section, we explain how we proceeded to select the replication paper, starting from the set of criteria determined. Then, we show the list of candidate studies.

4.1.1 Replication Selection Criteria

We defined the following criteria for selecting experiments for replication. CR1 stands for criterion 1, CR2 means criterion 2, and so on:

CR1: Papers published in conferences/journals about Software Engineering and HCSs.

CR2: Studies that can be replicated in an academic setting.

CR3: Studies that have at least 15 subjects or are executed in an environment without humans subjects.

CR4: Papers published after year 2000.

CR5: Papers that contain a clear description of subjects allocation and data analysis.

CR6: Papers with the replication package available (embedded in the paper or with a link to an external repository).

For **CR1**, a paper must have been published in Software Engineering conferences and journals which have HCSs studies as well. Specific conferences about HCSs, variability and related themes are also valid for this criterion. A reason for the usefulness of aiming at already published studies is because papers submitted in those conferences/journals have been extensively reviewed. Thus, we can expect experiments with good quality in terms of research design and reporting. The journals considered were: ACM Transactions on Software Engineering and Methodology (TOSEM), Automated Software Engineering (ASE), and Empirical Software Engineering (ESE). The considered conferences were: Systems and Software Product Line Conference (SPLC), International Conference on Software Engineering (ICSE), Empirical Software Engineering and Measurement (ESEM), Foundations of Software Engineering(FSE), International Conference on Software Reuse (ICSR), and Variability Modelling of Software-Intensive Systems (VaMoS), and Generative Programming and Component Engineering (GPCE).

CR2 considers that the study must be replicable in an academic environment. This criterion does not exclude experiments performed in the industry, but the candidate study must be replicable in an academic setting.

In the third criterion, **CR3**, we are aiming for experiments with a minimum of 15 subjects, since this is the number we initially estimated we would have in our operations (Chapter 5). An alternative situation is the experiment being executed in a computing environment (no humans subjects). In this case, we do not set a minimum subject number.

Also, we do not intend to replicate experiments originally performed before 2000 (**CR4**). For example, two of the reference books for Software Product Lines were published in 2001 (CLEMENTS; NORTHROP, 2002) and 2005 (POHL; BÖCKLE; LINDEN, 2005). If we consider that as a milestone for maturity of the concept of HCSs, it would be a sensible decision to not attempt to replicate experiments much earlier than that.

The fifth criterion, **CR5**, considers that the paper must contain a clear description of data analysis and subjects allocation. Although it may seem redundant, since the studies are taken from conferences and journals with a strong review phase, sometimes the empirical description is not the main focus of the paper.

Furthermore, it is highly desirable that an experiment has a published replication package (**CR6**). Having a replication package shows that the researchers of a baseline study wish that their experiment be replicated and provide means to perform it.

4.1.2 Candidate Papers

Using the criteria mentioned above, we started to search for a replication candidate in the journals and conferences described in the previous section. Our filtering started by inspecting manually each venue's proceedings starting from the year 2000 (criterion CR4) to 2018. Afterward, a search was performed on the papers' titles. We selected studies containing the terms Highly-configurable Systems, and replication (with their variations). From this phase, we obtained 68 papers (see Appendix 2).

After that, we read all 68 papers to elaborate on a final list. In that phase, we were able to identify which papers fit all criteria mentioned. We identified seven candidates with

Table 4.1 Candidate Papers for replication

Subjects Type	Venue	Paper	Reason For Declining
Computer	ESEC/FSE	Scalable Analysis of Variable Software	Replication assets not reachable due broken links.
	ESEC/FSE	Counterexample Guided Abstraction Refinement of Product-Line Behavioural Models	Replication assets not reachable due broken links.
	ESEC/FSE	Performance-Influence Models for Highly Configurable Systems	It might require up to 2 months to be executed. Our subjects would not have this time availability.
	ICSE	A Comparison of 10 sampling algorithms	-
	ICSE	Scalable prediction of non-functional properties in software product lines	It requires skilled subjects for interpretation of results.
Human	ESE	Do background colors improve program comprehension in the #ifdef hell?	Experiment requires skilled subjects.
	ESE	Comprehensibility of UML-based software product line specifications	Original study uses 116 subjects. We were not able to gather a similar amount of subjects.
	ICSE	How Does the Degree of Variability Affect Bug Finding?	Original study uses 96 subjects. We were not able to gather a similar amount of subjects.

accessible replication packages. We then grouped them based on the type of subjects involved with the experiments. The final candidate papers can be seen in Table 4.1, together with the issue that leads to excluding a paper from the candidate list. Finally, we ended up with one study from ICSE, named: "A Comparison of 10 sampling algorithms" (MEDEIROS et al., 2016).

4.2 GOAL AND RESEARCH QUESTIONS OF THE ORIGINAL STUDY

The objective of the original study was to compare 10 sampling algorithms for testing configurable systems. When a configurable system is developed, there is often the need to conduct test, similar to any other software. Testing HCSs adds an extra layer of complexity since many variations of an HCS can exist (NETO et al., 2011; MACHADO et al., 2014). Unique combinations of features might trigger some bugs that normally would not appear in an HCS with all variants present. However, it might be unfeasible to check every possible combination in a configurable system due to limited resources like time and computation power. Therefore, instead of testing all possible combinations, one can rely on sampling techniques which can then be used for performing software testing with satisfactory coverage.

Medeiros et al. (2016) investigated 10 state-of-the-art sampling algorithms and analyzed them in terms of fault-detection coverage and size of the sample sets. They also performed analyses on combinations of algorithms in order to find if applying one algorithm after another produces interesting results. This investigation was guided by the following group of Research Questions (RQs) (referred as **Study 1** in the original paper):

RQ1: What is the number of configuration-related faults detected by each sampling algorithm?

RQ2: What is the size of the sample set selected by each sampling algorithm?

RQ3: What combinations of sampling algorithms maximize the number of faults detected by each sampling algorithm?

However, as Medeiros et al. (2016) state, most of the papers concerning sampling algorithms in C/C++ do not consider: (1) header files, (2) configuration constraints, (3) build systems, (4) and global analysis when assessing the performance of the algorithms. Those studies make the assumption that ignoring those four factors does not impact bug detection and the size of the sample set. Considering each ignored factor as an assumption, the following group of research questions (referred as **Study 2** in the original study) investigate if those assumptions are valid:

RQ4: What is the influence of the four assumptions on the feasibility to perform the analysis for each sampling algorithm?

RQ5: What is the influence of the four assumptions on the number of faults detected by each sampling algorithm?

RQ6: What is the influence of the four assumptions on the size of the sample set selected by each sampling algorithm?

The algorithms under investigation were T-WISE: (with T ranging from two to six), MOST-ENABLED-DISABLED, ONE-ENABLED, ONE-DISABLED, RANDOM, and STATEMENT-COVERAGE. To assess how they performed, in the first part of the study (RQ1, RQ2, RQ3), the algorithms were applied on 24 C/C++ open source projects. On the second part (RQ4, RQ5, RQ6), only two subject systems were feasible to the algorithms. That happened because, in order to verify the influence that configuration constraints impose on the algorithms performance, it is necessary to have some sort of feature model or other asset that contains information about constraints. It was the case only for Linux and BusyBox.

4.3 RESULTS

The results for Study 1 are summarized in Table 4.2. RQ1 corresponds to the number of faults each algorithm is able to detect, while RQ2 corresponds to samples per file (which is the size of the sample set). The performance for selected algorithms combinations, which was under investigation of RQ3, is displayed on the last four rows.

Table 4.3 exhibits the results for Study 2. The empty cells shows which sampling algorithms could not be executed under a particular assumption lifted, thus answering RQ4. RQ5 corresponds to the number of faults detected and, lastly, and RQ6 is answered by the "Configs" columns for each assumption.

The authors published a replication package, containing the assets used on the experiment for replication purposes.[1].

[1] ⟨http://www.dsc.ufcg.edu.br/~spg/sampling/⟩

Table 4.2 Results of Study 1. SOURCE: (MEDEIROS et al., 2016)

Sampling Algorithm	Faults	Samples/File
Statement-coverage	90	1.3
Most-enabled-disabled	105	1.3
One-enabled	107	1.7
One-disabled	108	1.7
Random	124	2.6
Pair-wise	125	1.8
Three-wise	129	2.5
Four-wise	132	3.7
Five-wise	135	6.0
Six-wise	135	10.0
Pair-wise and one-disabled (C1)	131	3.5
One-enabled, one-disabled, and statement-coverage (C2)	132	4.8
One-enabled, one-disabled, and most-enabled-disable (C3)	133	4.8
One-enabled, one-disabled, and pair-wise (C4)	134	5.4

Table 4.3 Results of Study 2. SOURCE: (MEDEIROS et al., 2016)

Algorithms	Constraints			Global analysis			Header Files			Build System		
	Faults	Configs	Rank	Faults	Configs	Rank	Faults	Configs	Rank	Faults	Configs	Rank
Pair-wise	33 ↓	30	5	—	—	—	39 =	936	4	33 ↓	2.8 ↑	4
Three-wise	—	2	—	—	—	—	43 =	1,218	5	42 ↓	3.9 ↑	5
Four-wise	—	—	—	—	—	—	45 =	1,639	7	45 =	5.7 ↑	8
Five-wise	—	—	—	—	—	—	—	—	—	47 =	8.3 ↑	9
Six-wise	—	—	—	—	—	—	—	—	—	47 =	12 ↑	10
Most-enabled-disabled	23 ↓	1.4 =	1	27 =	1.4	1	27 =	1.4 =	1	26 ↓	1.4 ↑	2
One-enabled	30 ↑	1.1 ↓	3	31	7,943	3	31	890	6	20 ↓	2.3 ↑	7
One-disabled	38 ↓	1.1 ↓	4	39 =	7,943	2	39 =	890	3	39 =	2.3 ↑	3
Random	39 ↓	4.1 =	6	29	8,123	4	40	17.2	2	41 =	4.2 ↑	6
Stmt-coverage	32 ↑	4.1 ↑	2	—	—	—	—	—	—	25 =	1.3 ↑	1

Some algorithms do not scale, indicated using dashes (-). We use, and to represent small changes in the number of faults and Size of sample set, as compared to our first study and we use an arrow to represent larger changes.

4.4 CHAPTER SUMMARY

In this chapter, we have presented the baseline study used for replication in our empirical study. We unveiled the process of selecting the paper from a list of candidates, based on specific selection criteria. Then, we described the original study design, research questions, and results. In the next chapter, we present the operations of our study, which includes multiple replications of the experiment shown above.

Chapter 5

EMPIRICAL STUDY OPERATIONS

This chapter describes the two operations of this dissertation's empirical study (Section 5.1 and Section 5.2). These operations follow the design established on Chapter 3. In each one, we explain the participant selection process and outline their profile. Further on, we describe the experiment replications and the focus groups sessions applications. Later, we show how data analysis was applied (Section 5.3).

5.1 FIRST OPERATION

The first operation was applied within a post-graduate course of Empirical Software Engineering. The course is offered every semester at Federal University of Bahia (UFBA), Brazil. The syllabus of this course include various topics related to Empirical Software Engineering, such as designing a Systematic Review, performing statistics analysis and developing experiments in Software Engineering. Fifteen students participated in this operation within a time period of three weeks.

5.1.1 Participants Selection

In the first instance, the students enrolled in the course of Empirical Software Engineering participated acting as researchers. This mentioned course is composed of students with different levels of expertise in Software Engineering. On the one hand, we have skilled and experienced software engineers, and on the other, we have alumni recently introduced to the Software Engineering topic. We consider these participants to be suitable because they were having close contact with Empirical Software Engineering, where they were just introduced to experiment design.

Regarding arranging participants in groups, a characterization form was developed (Form 1), so the participants would answer it and later they would be allocated in groups. This form intended to gather information such as, but not restricted to: participants' academic degree, experience with programming, experience with experiments, and knowledge about HCSs. The participant's distribution then was designed in a manner

Table 5.1 Participants Characterization - First Operation

Group	Name	Degree	Exp. with programming	Experience with Experiments	Knowledge with HCS
Group 1	Odin	Masters Student	More than 5 years	Participant	Has only basic notions about HCS
	Thor	Masters Student	More than 5 years	None	Has only basic notions about HCS
	Ymir	Graduate	Between 1 and 2 years	Participant	Has theorical knowledge
Group 2	Saturn	Graduate	More than 5 years	Participant	Has theorical knowledge
	Mercury	Masters Student	More than 5 years	Participant	Has theorical knowledge
	Helios	Graduate	Between 1 and 2 years	None	Never heard before
Group 3	Venus	Masters Student	No experience	None	Has only basic notions about HCS
	Mars	Master	Between 1 and 2 years	None	Only heard about
	Jupiter	Ph.D Student	More than 5 years	None	Has only basic notions about HCS
Group 4	Neptune	Undergrad Student	Between 2 and 5 years	None	Theoretical and practical experience
	Pluto	Graduate	More than 5 years	Participant	Has only basic notions about HCS
	Minerva	Masters Student	No experience	Participant	Has only basic notions about HCS
Group 5	Vulcan	Masters Student	Between 2 and 5 years	None	Only heard about
	Diana	Masters Student	Between 2 and 5 years	Participant	Theoretical and practical experience
	Bacchus	Masters Student	Less than 1 year	None	Has only basic notions about HCS

that researchers with different expertise could be placed across the groups. For instance, we allocated, in the same team, a participant with reasonable experience in programming together with an experienced researcher. Contrasting with that, there was another group formed entirely by seasoned developers. By distributing researchers with different expertise, we intended to bring diverse points of view regarding each phenomenon. This diversity is highly desirable for performing axial coding in the constant comparison method (used in this study for qualitative analysis, Subsection 3.5.1), in which the most diverse variations of a phenomenon are found, the better description of a phenomenon there will be (STRAUSS; CORBIN, 1998).

The participants characterization can be seen on Table 5.1. The real names were omitted for confidentiality reasons. The activities performed by the participants can be seen in Figure 5.1. Groups 1 to 5 correspond solely to the first operation of the study.

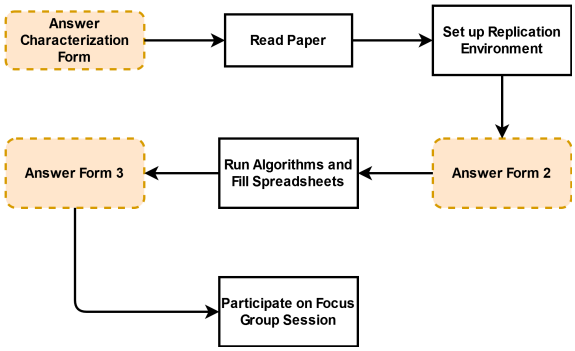

Figure 5.1 Participants activities

5.1.2 Experiment Replications and Focus Group

After the participants' allocation, on the first encounter between the author of this dissertation and the participants, there was an explanation of the basic concepts necessary to understanding the experiment. Since on this operation many participants were not

familiar with HCSs, and the time given to read the paper was relatively short, we considered being important to lecture them on this topic. The participants were given one week to read the original experiment paper.

On the second encounter, after all the participants read the paper, the author clarified questions regarding the original study. Although this might have introduced bias (see Threats to Validity), we judged this to be useful due to the lack of familiarity some participants had with the themes addressed in the study. Later, the author gave students one week to set up the environment and to answer Form 2.

On the next week, there was an additional encounter. Considering that the participants had configured their environments to replicate the experiment, the author gave instructions on how the experiment results would be documented. Thus, the participants would have one week to execute the experiment and insert their obtained results on the extraction spreadsheet. At the end of the activity, the participants would respond to Form 3, which had high-level questions about replication difficulties.

After all the groups finished performing the replications, the focus group sessions were performed at UFBA, with each team alternately. We reserved rooms to provide a quiet place for the sessions. Thus, we were able to produce audio recordings with as much intelligible sound as possible. Another action to minimize the amount of unintelligible audio was placing one of the audio recorders close to the moderator and the other one close to the participants.

5.2 SECOND OPERATION

In the same university, there is a research lab focused on Software Reuse named RiSE (Reuse in Software Engineering). The second operation was performed six months after the first iteration with nine masters and Ph.D. students from this laboratory. In total, each group had 2 weeks to perform the replications.

5.2.1 Participants Selection

Contrasting with the first operation, in the second instance of the study, we had access to researchers familiar with research in Empirical Software Engineering and HCSs. Thus, this time we have selected a more experienced group of participants to collect a different view on replication difficulties. The participants characterization can be seen on Table 5.2. Groups 6 to 8 represent teams participating on the second operation.

Although all participants from the second operation were members of the RiSE research group, in Group 8 the participants were not in Salvador, Brazil, at the period the empirical study was applied. Poseidon (names depicted here are fictional) was in England, Persephone was in the United States, while Zeus was in another Brazilian city. Therefore, their group was allowed to perform the study remotely.

5.2.2 Experiment Replications and Focus Group

As mentioned before, the second operation was applied within a time frame of two weeks. Since on this operation the participants were experienced with the themes addressed, we

Table 5.2 Participants Characterization - Second Operation

Group	Name	Degree	Exp. with programming	Experience with Experiments	Knowledge with HCS
Group 6	Athena	Masters Student	Between 2 and 5 years	Designed 3 to 5 experiments	Theoretical and practical experience
	Artemis	Masters Student	Between 2 and 5 years	Participant	Theoretical and practical experience
	Dionysus	Ph.D Student	Between 2 and 5 years	Designed 3 to 5 experiments	Theoretical and practical experience
Group 7	Cronus	Ph.D Student	Less than 1 year	Designed 2 experiments	Theoretical and practical experience
	Apollo	Masters Student	More than 5 years	Participant	Theoretical and practical experience
	Hestia	Ph.D Student	Between 1 and 2 years	Designed 2 experiments	Has only basic notions about HCS
Group 8	Zeus	Ph.D	More than 5 years	Designed above 5 experiments	Theoretical and practical experience
	Poseidon	Ph.D Student	More than 5 years	Designed 2 experiments	Theoretical and practical experience
	Persephone	Ph.D Student	More than 5 years	Designed 2 experiments	Theoretical and practical experience

judged that it would not be necessary to provide training on that matter. Therefore, the operation could be shortened by one week. In this operation, there was only one encounter between the author of this dissertation and the participants (excluding the focus group session that would happen later).

From email, the author explained the activity the participants would perform on the following week: read the paper. After one week, the author and participants had their encounter. The author of this dissertation answered questions about the paper and explained the further steps of the empirical study. The participants had one week to set up the environment and replicate the experiment. After executing the baseline experiment, the participants filled the extraction spreadsheets and returned them to the author.

Similar to the first instance, on the second operation of the study, after each team performed the activities related to the replication, the focus group sessions were conducted to gather feedback from the participants. Groups 6 and 7 had their sessions at UFBAs rooms scheduled for this purpose, with the same approach used in Groups 1 to 5. For Group 8, the focus group was conducted through a virtual meeting. Fortunately, the online session happened without major difficulties.

5.3 DATA ANALYSIS OPERATION

After all sessions were conducted and recorded, they were transcribed into text format. On top of those transcriptions, we applied the qualitative data analysis, as described in Section 3.5. We have used the tool QDA Miner Lite[1] for labeling the text. An example of a text excerpt in QDA Miner can be seen in Figure 5.2 (in Portuguese).

In the open coding process, we split the codes into two sets: the ones describing replication problems (which we name here as central codes) and the ones related to other phenomena that help explaining replication problems (called periphery codes). For instance, we have a code called "Difficulty in setting the environment up". In that code, we have a description of the difficulties the user experienced in setting up the environment. One reason that might be linked to that difficulty is the lack of ability with Linux (to set the environment, a participant has to install and configure a Linux distribution). So, the periphery code helps to explain the central code.

After collecting all text excerpts, in the axial coding we synthesized the explanations regarding the phenomena identified. In other words, after reading all the testimonials

[1]Available in ⟨https://provalisresearch.com/products/qualitative-data-analysis-software/freeware/⟩

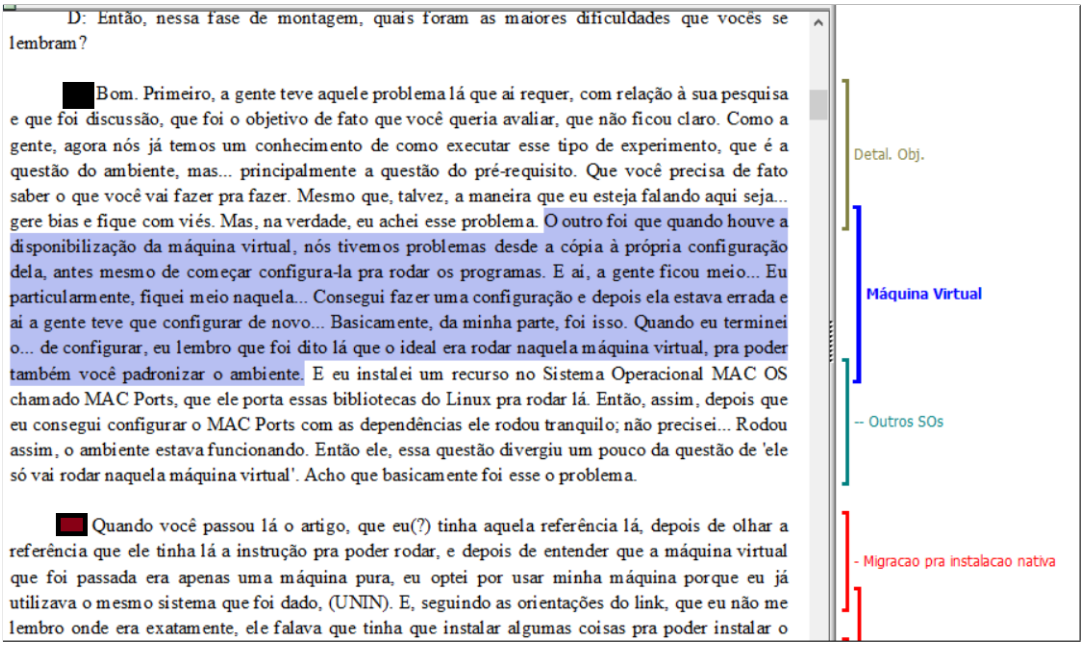

Figure 5.2 Open coding example as seen in QDA Miner

about each code, we structured them logically into flowing text. Our idea was to use the concepts from the paradigm and then elaborating on extensive explanations. An example of a synthesized explanation coming from the axial coding can be seen in Figure 5.3 (in Portuguese, the participant's citations, and in English, the code synthesis).

5.4 CHAPTER SUMMARY

In this chapter, we explained how we performed the two operations of the empirical study: the participants' selection, experiment replication, and focus group. Additionally, we explained the data analysis operation as well. In the next chapter, we present the results emerged from the operations performed.

- **Extração do code.zip (Addressed by 3 teams)**

Some teams had difficulties extracting code.zip file, which was the file containing the source codes of the projects. That happened because when setting the configuration up of the virtual machine, Virtual Box allocates by default 10 GB as a limit to the virtual Hard Drive Size. However, after installing Ubuntu and extracting the folder, this hard drive's size was not sufficient to store the extracted files. (Porque quando descompactava, ficava maior que o HD da máquina virtual.)
(Porque a gente tentava rodar com "code.zip", tentava baixar internamente, então tentava desinstalar coisas dentro do sistema pra poder arranjar espaço e não conseguia.) (Tamanho da VM)
So, the extraction operation made the Virtual Machine to freeze for one team,
(Era o tamanho da máquina virtual. Aí quando ia extrair o "code.zip", ele travava.)
and for another made a participant to lose a Virtual Machine (in his words "broke the image").
(Fiquei com algumas limitações de tamanho e tudo mais, aí acabou quebrando a máquina virtual. Eu perdi essa, a imagem. Tive que fazer um import da máquina virtual do zero porque... Não sei se foi o code [code.zip], alguma coisa assim, no conjunto de projetos que acabou estourando.)

A participant even suggested that the issues with the extractions might have impacted the replication quantitative results, however he wasn't able to explain clearly the reason and he wasn't sure of that either. (Diferença pequena. A gente não sabe exatamente o que causou. Mas era uma diferença pequena. Eu suponho que pode estar relacionado ao uso do "code.zip" ou não. Eu suponho. Porque o tamanho da pasta que estava dentro do experimento era menos, então pode ter uma relação. Mas eu não tenho certeza.)

Figure 5.3 Axial coding example

Chapter 6

RESULTS

We present in this chapter the quantitative (Section 6.1) and qualitative analysis results (Section 6.2). The qualitative analysis results describe mainly the replication problems found and additionally other phenomena surrounding the replications. The quantitative analysis results refers to the numbers obtained the in the replications.

We also describe an additional investigation we performed about teaching in Empirical Software Engineering (Section 6.3). On top of the results and the additional investigation, we provide recommendations (Section 6.4) and list threats to validity (Section 6.5).

6.1 QUANTITATIVE ANALYSIS

In this section, we present and discuss the results obtained from the sampling algorithms executed by the teams on the replication.

6.1.1 Study 1 results - Ignoring limiting assumptions

This subsection contains the algorithms execution replication results. The results can be seen in Table 6.1 (for bugs) and Table 6.2 (for configuration per file). The results of the baseline study are on the first column (Baseline).

For STATEMENT-COVERAGE algorithm, only Groups 5 and 6 were able to obtain results. According to Jupiter[1] from Group 2, the reason is that the algorithm took a long time to execute without giving any progress information. The few teams which found results got converging numbers compared to the baseline for this algorithm.

Regarding MOST-ENABLED-DISABLED algorithm, except for Group 3, all teams were able to find the same result as the baseline. Group 5 said this algorithm was not present in the replication artifacts. We assume that they were not confident enough to associate all-enabled-disabled (which was the name used in Java implementation) to most disabled.

[1]Table 5.1 and Table 5.2 show the participants allocation. Actual names are omitted for confidentiality issues.

In comparison with STATEMENT-COVERAGE algorithm, RANDOM algorithm first yields actual output shown on the console, but it was not considered meaningful enough for the participants. Once again, the lack of feedback might have discouraged participants to keep waiting until the algorithm was ran successfully. From all the teams, Group 3, 5 and 6 were able to execute this algorithm until the end. Group 3 only found number of bugs, and Group 5 and 6 found results for bugs and configurations per file for RANDOM. Due the nature of this particular algorithm, we should not expect to find the same result as the baseline. However, the value for Configuration per File of Group 5 (Table 6.2) seems to be way too dissonant from all the other values found for the other algorithms. We believe that something went wrong in the execution of this algorithm for this team and therefore this result should not be considered. Another unique case was the fact that Group 6 found matching results compared to the original study.

Apart from Group 3, all the teams were able to find converging results compared to the original paper for the remaining algorithms: ONE-ENABLED, ONE-DISABLED and T-WISE sampling algorithms.

For algorithms combinations, teams were able to find similar numbers for the bugs and completely different numbers for Configurations per Sample. The results can be seen in Tables 6.1 and 6.2. For combinations C1, C2 and C4, in general, the teams found the number of bugs being added of one compared to the baseline. So, for C1, while the baseline's result was 131 bugs, teams found 132, except for Groups 3, 6, and 8, which found 128. In C2, the baseline amount of bugs was 132, while the teams found 133, except group 7, which found 112 bugs. C4, on the baseline contained 134 bugs, while the groups found 135 as a result, except Group 7 which found 129.

Regarding algorithms combinations results for Configurations per Sample, none of the teams were able to find similar results compared to the baseline. When running Java classes responsible for retrieving these numbers, the output actually does not have results for Configurations per File (although the original paper has results for that.) Groups 2, 3, 4, and 8 did not deliver any results for Configurations per Sample for the algorithms combination. Groups 1, 5 and 7 inserted on the table the configurations number instead (the Java implementation only gave results for bugs and configurations, but not Configuration per File). Group 6 attempted to calculate the results (since the total number of project files was 50078). For instance, taking the combination C1 if 2803 (number of configurations given by the algorithms) is divided by 50078 (total number of files) the answer is approximately 0.06, which was the number Group 6 placed on the Table. However, this number does not confirm the baseline. In the end, we can state that for configuration per file, none of the groups were able to replicate the baseline.

6.1.2 Study 2 results - Lifting limiting assumptions

In this subsection, we discuss the results for the Study 2, where the assumptions are lifted in order to perform a more realistic bugs calculation. The following assumptions were lifted: Configuration Constraints, Global Analysis, Header files, and Build System. The results can be seen respectively in Table 6.3, Table 6.4, Table 6.5, and Table 6.6. A dash in the cell indicates that the algorithm execution is not feasible for that assumption lifted

Table 6.1 Study 1 Replication Results - Bugs

Sampling Algorithm	Baseline	G1	G2	G3	G4	G5	G6	G7	G8
Statement-coverage	90	-	-	-	-	90	90	-	-
Most-enabled-disabled	105	105	105	-	105	105	105	105	105
One-enabled	107	107	107	107	107	107	107	107	107
One-disabled	108	108	108	108	108	108	108	108	108
Random	124	-	-	81	-	134	124	-	-
Pair-wise	125	125	125	125	125	125	125	125	125
Three-wise	129	129	129	129	129	129	129	129	129
Four-wise	132	132	132	132	132	132	132	132	132
Five-wise	135	135	135	135	135	135	135	135	135
Six-wise	135	135	135	135	135	135	135	135	135
C1	131	132	132	128	132	132	128	132	128
C2	132	133	133	133	133	133	133	112	133
C3	133	133	133	-	133	133	133	133	-
C4	132	135	135	135	135	135	135	129	135

Table 6.2 Study 2 Replication Results - Samples per File

Sampling Algorithm	Baseline	G1	G2	G3	G4	G5	G6	G7	G8
Statement-coverage	1.3	-	-	-	-	1.3	1.3	-	-
Most-enabled-disabled	1.3	1.3	1.3	-	1.3	1.3	1.3	1.2	1.3
One-enabled	1.7	1.7	1.7	2.71×10^{15}	1.7	1.7	1.7	1.7	1.7
One-disabled	1.7	1.7	1.7	$.71 \times 10^{15}$	1.7	1.7	1.7	1.7	1.7
Random	2.6	-	-	-	-	186578	2.6	-	-
Pair-wise	1.8	1.8	1.8	2.75×10^{16}	1.8	1.8	1.8	1.7	1.8
Three-wise	2.5	2.5	2.5	3.43×10^{15}	2.5	2.5	2.5	2.4	2,5
Four-wise	3.7	3.7	3.7	4.71×10^{15}	3.7	3.7	3.7	3.7	3.7
Five-wise	6.0	6	6	6.94×10^{15}	6	6	6	5,9	6
Six-wise	10.0	10.0	10.0	$\times 10^{16}$	10.0	10.0	10.0	10.0	10.0
C1	131	2803	-	-	-	2.8	0.06	2803	2803
C2	132	4662	-	-	-	4662	0.09	907	4662
C3	133	4126	-	-	-	4731	0.08	4126	-
C4	132	4731	-	-	-	4731	0.09	1145	4731

Table 6.3 Study 2 Replication Results - Constraints

Sampling Algorithm	Baseline	G1	G2	G3	G4	G5	G6	G7	G8
Statement-coverage	32	32	32	-	-	32	32	-	32
Most-enabled-disabled	23	23	23	-	23	23	23	23	23
One-enabled	30	7	7	7	7	7	7	7	7
One-disabled	38	6	6	6	6	6	6	6	6
Random	39	-	-	-	-	42	44	-	-
Pair-wise	33	-	-	-	-	-	-	-	-
Three-wise	-	-	-	-	-	-	-	-	-
Four-wise	-	-	-	-	-	-	-	-	-
Five-wise	-	-	-	-	-	-	-	-	-
Six-wise	-	-	-	-	-	-	-	-	-

or the group left the field empty for any other reason.

For the Constraint-enabled scenarios, among the feasible algorithms, PAIR-WISE was the only one in which no team was able to retrieve any results. In MOST-ENABLED-DISABLED, all teams except Group 3 confirmed the baseline, with 23 bugs. For ONE-ENABLE and ONE-DISABLED algorithms, all the teams found the same result for each algorithm: 7 bugs for ONE-ENABLED and 6 bugs for ONE-DISABLED, while the baseline yields 30 bugs for ONE-ENABLED and 38-bugs for ONE-DISABLED (which can be considered a significant discrepancy). In STATEMENT-COVERAGE case, the four groups which filled the cells for this lifted assumption found 32 bugs, confirming the original result of 32 bugs. Only two groups were able to run RANDOM: while the baseline result was 39 bugs, Group 5 found 42 and Group 6 found 44, which still can be considered an expected result due the nature of random sampling.

Regarding the results for Global Analysis, T-WISE and STATEMENT-COVERAGE were not supposed to be executable. However, all teams but Group 3 were able to retrieve results (being 23 bugs). For the feasible algorithms, ONE-ENABLE presented the same value as the original paper, 31 bugs. For ONE-DISABLED, while the baseline found 39 bugs, all teams could get close to that, which was 38. A similar situation occurred in ONE-DISABLED, where all teams found 38 bugs, almost reaching the 39 bugs of the reference value. The other algorithms showed discrepancy: MOST-ENABLED-DISABLED yielded 35 bugs on the replication for all groups but Group 3 (which could not run the algorithm) contrasting with 27 bugs from the baseline; and RANDOM replicated values orbited around 38.

In terms of the Header Files constraint, STATEMENT-COVERAGE, five and six-wise did not scale. We saw a similar situation in Global Analysis. MOST-ENABLED-DISABLED baseline was 27 bugs, against 35 which all groups except Group 3 and 8 obtained. THREE-WISE and FOUR-WISE followed closely the baseline values: for the former 45 bugs against 43 bugs from the baseline; for the latter 46 bugs against 45 bugs. RANDOM orbited

Table 6.4 Study 2 Replication Results - Global Analysis

Sampling Algorithm	Baseline	G1	G2	G3	G4	G5	G6	G7	G8
Statement-coverage	-	0	0	0	0	0	-	0	0
Most-enabled-disabled	27	35	35	-	35	35	35	35	35
One-enabled	31	31	31	31	31	31	31	31	31
One-disabled	39	38	38	38	38	38	38	38	38
Random	29	36	39	37	40	39	38	38	38
Pair-wise	-	23	23	-	23	23	23	23	23
Three-wise	-	-	-	-	-	-	-	-	-
Four-wise	-	-	-	-	-	-	-	-	-
Five-wise	-	-	-	-	-	-	-	-	-
Six-wise	-	-	-	-	-	-	-	-	-

Table 6.5 Study 2 Replication Results - Header Files

Sampling Algorithm	Baseline	G1	G2	G3	G4	G5	G6	G7	G8
Statement-coverage	-	0	0	-	0	0	31	31	0
Most-enabled-disabled	27	35	35	-	35	35	35	35	-
One-enabled	31	31	31	31	31	31	31	31	-
One-disabled	39	39	39	39	39	39	39	39	-
Random	40	47	43	46	44	44	42	46	45
Pair-wise	39	39	39	39	39	39	39	39	39
Three-wise	43	45	45	45	45	45	45	45	45
Four-wise	45	46	46	46	46	46	46	46	46
Five-wise	-	-	-	-	-	-	-	-	-
Six-wise	-	-	-	-	-	-	-	-	-

Table 6.6 Study 2 Replication Results - Build System

Sampling Algorithm	Baseline	G1	G2	G3	G4	G5	G6	G7	G8
Statement-coverage	25	31	31	31	31	0	31	31	0
Most-enabled-disabled	26	28	28	-	28	28	28	28	35
One-enabled	20	20	20	20	20	20	20	20	31
One-disabled	39	39	39	39	39	39	39	39	38
Random	41	42	45	41	40	38	41	42	38
Pair-wise	33	33	33	33	36	33	33	33	33
Three-wise	42	46	46	46	46	46	46	46	-
Four-wise	45	45	45	45	45	45	45	45	-
Five-wise	47	47	47	47	47	47	47	47	-
Six-wise	47	47	47	47	47	47	47	47	-

around 45 bugs, against 40 bugs in the original. The other algorithms matched the baseline, excluding Group 8 in ONE-ENABLED and ONE-DISABLED which did not get any results.

Considering Build System constraint, results matched the baseline for the following algorithms: ONE-ENABLE; ONE-DISABLED; PAIR, FOUR, FIVE and SIX-WISE, and RANDOM (taking the average from all groups results). STATEMENT-COVERAGE original's number of bugs was 25, against 31 bugs found by most of the groups. MOST-ENABLED-DISABLED result on the baseline was 26, closely followed 28 bugs which the majority of the teams found. For THREE-WISE, while most of the teams found 46 bugs, the original value was 42. It is important to notice that all group obtained similar values in all algorithms, except Group 8 which got diverging results for all algorithms but RANDOM.

6.2 QUALITATIVE ANALYSIS

After the focus group sessions, the audio files were transcribed in order to perform qualitative analysis (constant comparison) on them. In this section, we present the outcomes of this step.

6.2.1 Open coding

During the open coding, a total of 96 codes were generated. Later, the codes were grouped in higher-level categories, such as: environment set up, algorithms execution, paper interpretation, participants replication experience (which are the questions sections of the Focus Group), comments on the virtual machine, participants' attitude, participants' complaints, and miscellaneous (codes not related to any of the previous categories). After the individual labeling, two researchers calculate the inter-rater reliability and found a Cohen's Kappa of 0.88. Cohen's kappa is a statistic that assess the degree of agreement between the codes assigned by two researchers working independently on the same

sample (NOLL; BEECHAM; SEICHTER, 2011). Values of Cohen's kappa fall between 0 and 1, where 0 indicates poor agreement, and 1 perfect agreement. According to the thresholds proposed by Landis and Koch (1977), our kappa value of 0.88 indicate almost perfect agreement among raters.

6.2.2 Axial coding

During the Axial coding phase, we set "Replication Difficulties" as a central phenomenon (explained in Section 3.5.1). Then, we reallocated the codes under three branches, on which we perform the paradigm. It means that for each branch, we proceed to find: the causal events, the actions participants execute on those events, and the consequences of those actions. The branches, showed in Figure 6.1, are the following: difficulties in setting up the environment, difficulties in algorithms execution, and difficulties in interpreting the paper (which match the correspondent categories created during open coding).

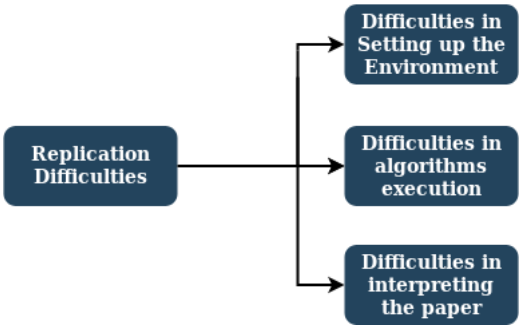

Figure 6.1 Axial coding paradigm branches

6.2.2.1 Setting up the Environment Throughout the coding, twelve difficulties emerged related to the phase of setting up the environment. Those problems can be grouped into the following groups: Projects extraction, Dependency problems and Problems with software versions.

Projects extraction difficulties involve placing the projects to be analyzed by the sampling algorithm on the appropriate location. It includes downloading the projects from the supplementary website, extracting the compressed folder and importing them into the Java project.

Group 1 judged this task of placing the projects folder on the appropriate location to be straightforward (Odin: "And there was the detail that it was required to put the folder code.zip within the [eclipse] project and that was not complicated"). However, other teams missed having more descriptive instructions, like where the folder should be placed or imported to the eclipse project containing the algorithms. On the other hand, Group 2, in particular, mentioned a trial-and-error approach until they could find the appropriate location. (Saturn: "I think the author could detail more specifically to which folder to put the C projects. Then sometimes I put the projects in a place which gave

me an error. Then later, as Mercury was able to execute the algorithms, he said: 'No, it's the other folder here'.")

Dependency problems are formed by replication difficulties related to dependencies necessary to run the algorithms. In general, the teams complained that the libraries versions were not described on the supplementary website. Another cited problem is that there was a dependency which was not included on the instructions (named "flex"), so whether the participants figured out themselves or required assistance to this dissertations' author to overcome this difficulty.

One dependency, called "undertaker", was often cited by the participants as being troublesome. The baseline paper used a modified version of this library, so in order to install this modified version it was required to install this package from the sources (which was how it was described on the supplementary website). However, many participants were not proficient in compiling packages from scratch in Linux. So when they faced compilation problems, they installed the version present in Ubuntu's repository. But this version was not the modified one provided by the paper. Thus, the teams which did not install from the source code should not be able to obtain results on STATEMENT-COVERAGE algorithm on the second study, constraints part (see Section 6.1.2).

At last, there were difficulties concerning software versions. Although the supplementary website contained instructions for setting up the environment and required software to install in order to execute the sampling algorithms, the participants stated in many occasions that their version was not described. The most cited required software missing a version were eclipse (since the algorithms were embedded in an eclipse project), Java and the Linux distribution. One team expressed the feeling, if those information were present, it would be easier to recreate the conditions for replications similar to the original. This team also proposed that it would be convenient to have a complete replication environment already set by the authors, with the OS, libraries and algorithms already configured and packed for running without further complications.

One team reported the difficulty of finding old working versions of dependencies required to run the experiment. Sometimes it's required to have old versions since they used Ubuntu 14.04 and the newer dependencies are not compatible with older versions of Ubuntu (unless you update a huge range of dependencies which is not the most elegant solution).

6.2.2.2 Algorithms Execution The difficulties found in the sampling algorithms execution phase were clustered in two categories: inconsistencies and abnormalities in execution. Inconsistencies refer to name in consistences happening around the terminology used around the paper itself, the supplementary website and the replication artifacts. Abnormalities refer to errors and failures found while executing the algorithms. There were also further comments made by the participants which do not fit in any category listed.

Group 1 reported that it was easy to identify the algorithms by the output, matching the results described on the paper in terms of algorithms name. However there were instances of differences of the algorithms names on the paper and the execution outputs/names on the source code. For instance, the algorithm on the baseline called

most-enabled-disabled is shown on the algorithm's output as all-enabled-disabled. Group 8 (which was composed by experienced researchers) recommended consistency between the names on the artifacts an on the paper (Zeus: "In summary, the nomenclature he [the author] has to use in the experiment is the same he uses on the [supplementary] website; it's the same that the paper uses. Therefore, that is the recommendation: use the same terms.")

Among the abnormalities, a couple stood out: execution failures and long wait time for some algorithms. Execution failures were mainly caused by a specific issue: at one line code there was a folder path making reference to the baseline's first author computer. Groups 1, 2, 6, and 8 mentioned this defect. Regarding the algorithms that took a long time to be executed, they were STATEMENT-COVERAGE on the first experiment and RANDOM, on first and second experiments. For STATEMENT-COVERAGE in the first study, Group 3 gave up on running the algorithm after wait for around one and a half hours. A common problem on that situation is that there was not feedback of the time estimated to run the algorithm or a progress indicator, which made the participants to wonder if the execution was running appropriately or not.

Another problem mentioned was that the algorithms execution output was not clear enough. Groups 5, 6 ,7 and 8 mentioned issues related to execution's output clarity. In Dionysus's (Group 6) opinion, the verbosity level was not adequate for the random algorithm (which, differently from most algorithms, showed additional information besides the execution that wasn't useful for a replication context). Zeus (Group 8) expressed a similar feeling towards this (Zeus: "On the last executions there, for instance, on the output we saw there were paths with configurations there were simply intermediary stuff, which he only used to make sure that the final configuration was the sum of the paths configurations. For whom is executing, it's just disturbing.")

6.2.2.3 Interpreting the paper On the phase of interpreting the paper, there were two major groups of difficulties: difficulties caused by the lack of familiarity with the themes addressed on the paper by the participants and difficulties regarding the convergence of results from the baseline. Problems with the convergence of results have been extensively discussed in Section 6.1. Therefore, the focus of discussion in this subsection will revolve around the lack of familiarity.

Regarding the difficulties caused by the lack of familiarity with the themes, there were mainly the following ones: understanding about replications, and insufficient academic background.

Concerning participants' understanding about replications, Group 2 reported that they were unsure about how much intervention on the source code they were allowed to do to not invalidate the replication. One participant stated that the Java projects should not be modified at all since the experiment should be reproduced exactly like the original. However, this statement is not precisely aligned with replication definitions established by existing studies (BALDASSARRE et al., 2014). This indicates that not being familiar with the definitions can impact the way these studies are conducted.

6.2.3 Qualitative Analysis Summary

Using the constant comparison method, we analyzed the focus groups sessions' transcriptions. This constitutes the qualitative analysis phase. We applied open and axial coding to discover relevant phenomena surrounding the replications performed.

From the codes obtained, we identified two categories embracing most of the codes: imprecision on replication instructions (grouping projects extraction, dependency problems, name inconsistencies, long wait time, and execution output) and defects on the software (grouping dependency problems, and execution abnormalities). This indicates that the majority of problems found by the participants are related to information about steps to execute the replication being absent or unclear and replication assets not being seamlessly working. Also, insufficient background with replications and the themes addressed in the experiment put some participants into difficulties (tagged in code lack of familiarity). This indicates that experience with replications and the area under study might contribute positively to the replication success.

6.3 ADDITIONAL INVESTIGATION ABOUT ESE TEACHING

We present in this section an investigation about education practices in Empirical Software Engineering and replications.

During the qualitative analysis, we could observe that many participants were not familiar with replications. This brought them difficulties when replicating the experiment (Subsection 6.2.2.3). Since most of them were graduate students taking their Masters or Ph.D., one might wonder if courses addressing Empirical Software Engineering include replications as a lecturing topic. This way, future researchers would have a more solid background when attempting to perform replications. Being introduced in this matter is useful also to bring replication awareness when designing empirical studies.

In this sense, we proceeded to perform an ad-hoc investigation to get a glimpse of Empirical Software Engineering teaching regarding replication. We sent emails to the Program Committee members of the International Symposium on Empirical Software Engineering and Measurement (ESEM 2020)[2], technical papers track. Our intention with that was to reach professors involved in impacting ESE research. We asked them if 1) they were teaching empirical software engineering currently and; 2) if they could provide their ESE course syllabus (for the professors which responded affirmatively the previous question). In fact, with the second question, we intended to know if the professors' ESE courses addressed the topic of replications. Instead of asking them directly, we requested their courses' syllabus to inspect if the theme was addressed.

In total, we mailed 70 researchers. From the emails sent, we received responses from 41 people. Whereas 15 professors teach Empirical Software Engineering, only 6 of them include the topic of replication.

Out of the 6 courses teaching replication, the approach took in each of them was following: two presented this matter within experiment design theme, one course dedicated replication as a topic on its own, one presented the subject in the statistical analysis of

[2] ⟨https://eseiw2020.di.uniba.it/esem_conf/technical-papers/⟩

an experiment, and the remaining case we had to rely on the contacting professor which said they addressed this topic. It is important to highlight that in two courses, there was also a special topic about replication packages, which is an essential tool to develop trustworthy replications (SOLARI; VEGAS; JURISTO, 2018).

Therefore, we identified from this ad-hoc inquiring that, in education terms, even with active and proficient professors performing research in empirical software engineering obtained from our set of responding teachers, this discipline is not widespread lectured. This topic is addressed by only 36%. Regarding replications, only 14% includes replication in their syllabi.

6.4 RECOMMENDATIONS

In the previous sections, we discussed our findings. On top of those findings and our experience with the empirical study, agreeing also with existing research, we provide the following recommendations for researchers (Subsection 6.4.1), practitioners (Subsection 6.4.2), and educators (Subsection 6.4.3).

6.4.1 Researchers

For the researchers replicating an experiment, it is highly beneficial to communicate with the original research team (MAGALHAES et al., 2015). This communication can minimize failures in replication results (GILES, 2006) but it must be done diligently to not introduce bias. We recommend that this communication be performed only in cases where the replication team feels important information is omitted (for example, a instruction step is not clear for the reader). Still, the independence between the original team and the researchers replicating the experiment must be preserved.

In the testimonials provided by the participants during the focus group sessions, there were several mentions of instructions not being clear enough (Thor: "I think it was missing a bit of detail about the environment he [the original study's author] ran [the algorithms].") This agrees with the existing literature, which mentions difficulties in transferring tacit knowledge (SHULL et al., 2002). We recommend to researchers developing experiments to be aware of the communicability of their instructions. These instructions can be made available in the form of an external website or even a video on a streaming platform. Nevertheless, following replication guidelines (KITCHENHAM et al., 2006; CARVER, 2010) is advisable to minimize communication issues.

Additionally, existing research in Empirical Software Engineering has been supporting the idea that experimental studies should provide replication packages (Basili; Shull; Lanubile, 1999). We suggest using, for instance, Solari, Vegas and Juristo (2018) proposal to structure a replication package, which, on their research, have shown that using a well-structured replication package can be beneficial in this direction. Similarly, following ACM standards[3] for artifacts is encouraged.

Furthermore, in case there is need to execute additional software on the replication, based on the evidence found in this research, we consider it is essential to list the hardware

[3] ⟨https://www.acm.org/publications/policies/artifact-review-badging#available⟩

specifications and operating system configuration in which the experiment was originally executed. If there is no space on the paper, we advocate that this information is made available in the supplemental material.

6.4.2 Practitioners

In the original paper, Medeiros et al. (2016) have included guidance for practitioners. They state that there is no optimal sampling algorithm for every software project. They recommend using sampling algorithms with small sample sets when dealing with projects on their initial phases, so one could retrieve configuration faults quickly while the project is under deep changes and fast growth. When the software is larger and coming closer to release date, they recommend using algorithms with more comprehensive sample sets. However, under realistic scenarios (taking constraints, global analysis, header files and build system into consideration), many algorithms do not scale. Therefore, it is advisable using simple algorithms (like MOST-ENABLED-DISABLED) on those situations. In our replication, although we have seen numeric differences, most groups were able to execute the same algorithms which the original authors were able to. The algorithms which did not scale on the baseline were not able to be executed on the replication either. Thus, we agree with the recommendations given by the original paper.

Experiment replications are useful to test a technology in-house before applying it on the company (WOHLIN et al., 2012). A company interested in a replication paper can double-check their results in order to verify if the evidence holds under the company context.

We believe that achieving efficient communication with original authors applies not only to academia but also to industry. A pilot might be useful to test the instructions and the quality of the replication package as well before performing a full replication. These instructions should be more practical than theoretical since the former usually is more relevant for companies. Furthermore, it might be preferable to use papers from venues that test the quality of their assets (usually indicated by a badge or a marking on the header of the paper's first page).

6.4.3 Educators

The first operation of the exploratory study was applied with students enrolled in a course of Empirical Software Engineering. Some participants in this round were having the first contact with experiment design. On the focus group, a participant reported they had difficulties to recognize the elements composing the design of an experiment, such as subjects, input and output variables, and problem statement (Minerva: "No, no. I did not think it was the way the professor explained. I was not able to see them there [the experiment design elements]"). Not always scientific papers have all these elements in a dedicated section. Somehow, the student was expecting to see all the items explicitly described in the literature.

Therefore, based on the findings of the qualitative analysis, we recommend to Empirical Software Engineering educators, when lecturing on courses regarding experiment design, that they provide practical training to their students, and not stay focused only

on the theoretical part. Many students we collected testimonials from expressed the wish to design an experiment themselves (which they did not have the chance on the course they were enrolled). This suggests that by allowing the students to have contact with empirical studies from the beginning can be more meaningful to students, and therefore, might bring better learning on this subject.

Additionally, we developed an ad-hoc investigation to prospect if Empirical Software Engineering and replication topics are being taught in post-graduate courses (Section 6.3). We mailed 70 active professors developing impacting research asking if they address this discipline on their universities. From the responding authors (41 researchers), 36% teach ESE and 14% include replication as a topic. Probably, those numbers might be way lower if we consider all professors working with Software Engineering in general, which would be a more realistic scenario.

In light of those findings, we incentive that Empirical Software Engineering is more widespread taught. We believe this movement must go along with the rise of empirical research in Software Engineering, and it consequently can foster the development of Software Engineering research to be more evidence-driven. We recommend also inserting the topic of empirical studies replication on the syllabus of Empirical Software Engineering courses. Since we have observed one of the difficulties our participants reported was related to their insufficient background on that matter, those courses should be able to educate students about replications. As a growing topic in academia, a researcher will likely have to deal with replications on their career, whether replicating a paper whether designing an easily replicable study.

6.5 THREATS TO VALIDITY

Although we designed our study aiming minimizing threats to validity, some are still present. In this section, we present the internal, external, conclusion, and validity threats we were not able to mitigate.

6.5.1 Internal Validity

One issue that might have influenced how the participants performed the replication is their motivation. Since they were performing this task as part of a post-graduation course and conducting research was not their primary goal, it is likely that the participants were not vested as much as the researchers of the original paper. This motivation might have led the participants to spend less effort. In order to minimize that, we added additional cells to the extraction spreadsheets, which correspond to the data algorithms do not provide. This forces the participants to reason about what the algorithms' output mean, instead of only "copying and paste" the results onto the cells.

Additionally, participants tend to spend minimal effort to get the activities done. The participants performed the empirical study activities within the context of a post-graduation course. Neither the experiment or any of its related activities may be part of their research. Then, their motivation to be part of the replication is expected to be lesser than the motivation of a researcher interested in Software Engineering or Configurable

Systems.

Another threat to internal validity is communication between teams. On training sessions, it was explicitly stated that the teams should not communicate between themselves about the replication. When teams exchange such information, one group can teach the other how to avoid specific difficulties in replication. However, ensuring complete prevention of teams information sharing was not possible.

6.5.2 External Validity

The replication difficulties found in this dissertation might not extend to other replications. However, we still consider the lessons and recommendations found based on the testimonials of participants in this study to be valuable. Most of the replication difficulties are related to the replication artifacts and instructions. Researchers intending to replicate an experiment might take the lessons learned from this study to prepare their replication package in a way that minimizes replication difficulties and increases the communicability of replication instructions.

6.5.3 Conclusion Validity

Since this study relies heavily on the grounded theory's constant comparison method, the testimonials interpretation relies on a certain degree of subjectivity. The mapping from evidence sources to conclusions might be not perfectly drawn. However, this can be considered something expected when using this technique. We also use quantitative data to triangulate.

Furthermore, the replication does not cover the task of generating the covering arrays. This part could not be replicated and we had to rely on the arrays provided with the Java project. In the end, we are not able to verify the trustworthiness of the arrays or how the authors constructed them.

6.5.4 Construct Validity

The participants had a short time period - 2 to 3 - to perform the replication. Besides, most participants were not researchers of the specific topic of the baseline paper. Those two factors might impact the participants' comprehensibility and how they performed the replications. In order to mitigate those, this dissertation's author provided assistance whenever the participants faced problem. This assistance was provided in the least intrusive way as possible to avoid introducing unintentional bias. The exact response given by the author was written down to make sure that same answer is given to all groups in case they had similar questions.

6.5.5 Chapter Summary

In this chapter, we presented the results for our empirical study. Both quantitative and qualitative results are shown, the former being the numbers obtained in the replications, and the latter being the coding descriptions obtained from open and axial coding. Fol-

lowing that, we presented an additional investigation we performed concerning teaching practices in Software Engineering. Based on the results found, we provided recommendations for researchers, practitioners, and educators. Furthermore, we addressed the threats to validity: internal, external, conclusion, and validity. In the final chapter, we present the conclusions, contributions, and future work of this dissertation.

Chapter 7

CONCLUSION

As Empirical Software Engineering (ESE) has been providing evidence for phenomena surrounding Software Engineering, this evidence should be validated by different independent researchers, and under different circumstances. However, there might be difficulties in the replication process which may discourage researchers or even impact the results. In the following sections, we present the summary of research contributions, the research outcomes, future work, and the conclusions of this dissertation.

7.1 SUMMARY OF RESEARCH CONTRIBUTIONS

In general, the research presented in this dissertation makes the following contributions to the area of ESE: it provides evidence about replication difficulties of experiment replications; it functions providing a double-check on sampling algorithms evidence for Highly-configurable Systems (HCSs) found from previous research; and introduces a research methodology where researchers are put as the subjects while they perform a replication. Each contribution is summarized as follows:

7.1.1 Evidence about replication difficulties

While there is a need for increasing replications in Empirical Software Engineering, few papers address directly this issue. This dissertation provides light on that matter. We collected reports from the researchers, which performed the replications and provided their point of view regarding the experience of replicating as a whole. There is evidence of difficulties through the entire replication process, from reading and understanding an original study, passing through the experiment replication, and then the interpretation of the results found. Therefore, this dissertation offers contributions to the research community, with rich evidence about replication difficulties in Software Engineering, which is currently scarce.

7.1.2 A double-check on sampling algorithms evidence

Testing on HCS is challenging due to the variable nature of these kinds of systems. One strategy to mitigate complexity in testing HCS is using sampling algorithms. The study developed by Medeiros et al. (2016) present a comparison of 10 sampling algorithms for configurable-systems. With this study, an interested reader can select which of the assessed algorithms is more appropriate for their needs based on evidence. Our study provides a double-check on the results found on the baseline. In the end, we verified which of the algorithm's execution could be reproduced, and then we give back to the community an additional verification of the results exhibited on the original paper.

7.1.3 Research methodology

In the ESE community, we often find surveys targeting researchers. Thus, it is possible to investigate how researchers think and what they experience when developing empirical studies. However, different from that, in our study, we present a methodology in which researchers are on focus. Treating the researchers as subjects enables us to gather abundant feedback from them. Throughout the experiment replications, the application of the focus group sessions, and analysis using the constant comparison method, the empirical study orbits around the outputs provided by the participant researchers.

7.2 RESEARCH OUTCOMES

Besides the dissertation, there were other outcomes from the study. These are the research products coming from the empirical study:

- **Replication Package -** The importance of having a replication package has been emphasized on many occasions throughout the dissertation. The artifacts for this empirical study can be obtained in the GitHub's repository. There can be found the forms applied, the training material, and links to the original paper (and its respective replication package), virtual machine, and the operating system used.

- **Research Paper -** A paper with the research described in this dissertation has been produced and submitted to the journal Empirical Software Engineering, a high-impact venue. Until the closing of this dissertation, the article was under evaluation.

- **Positive Influence on the participants -** On the focus group sessions, the participants expressed to have experienced several difficulties, and even sometimes, showed discontent in participating in the study. However, in the end, many participants reported that participating in the replications was positive because: 1) it raises awareness of the importance of replication studies; and 2) the participants were proven that it is essential to have clear instructions and working artifacts for a replication to be successful. Thus, although it is difficult to guarantee that the participants are going to apply those experiences to future studies, it is likely that the researchers will be more conscious when designing and applying empirical studies.

7.3 FUTURE WORK

We can consider research focusing on replication difficulties in Software Engineering to be scarce. While this dissertation contributes with a study report replication difficulties from the researchers themselves, there is opportunity to further enrich the findings from this dissertation with the following future work:

- **Replicate Papers with the ACM Badge** - As future work, we intend to perform external replications on papers possessing the ACM badge (meaning that their assets have been tested and certified to be working). We could evaluate if, by receiving the certification the artifacts are reusable, this will necessarily translate in replication with fewer replication problems and difficulties.
- **Replications with Human Subjects** - We intend to perform also experiment replications with human subjects to investigate difficulties in this matter. We assume that empirical studies that use human participants on their design can exhibit different challenges. As a matter of comparison, in our study, we used human participants only in the upper empirical study, but the experiment itself was executed entirely on the computer.
- **Survey with Researchers Performing Replications** - Another possibility for future work would be conducting a survey with researchers who perform replications in Software Engineering. This way we could capture challenges and difficulties the research community actually experience, and then we could map issues impacting current replications in empirical research.

7.4 CONCLUDING REMARKS

We conducted an exploratory study aiming to understand which replication difficulties might emerge when replicating a Software Engineering experiment, and more specific, a study involving HCSs.

We have developed a research methodology in which the participants emulated researchers replicating the experiment. We had in total eight replications, performed by groups of three people each, obtaining results and providing testimonies concerning the replication. On top of that, we collected quantitative data (numeric results from the replications) and qualitative data (codes coming from constant comparison method application).

Our results show that there are four categories of challenges pertaining to replication: Setting Up the Environment, Algorithms Execution, Paper Interpretation and Replication Experience (which intersects with all categories before). From the participants' testimonials we found that most of the difficulties can be traced back to lack of preparedness of the replication artifacts and clarity of the instructions. Additionally, during the axial coding, we could observe that most codes describing difficulties overlapped codes related to those two issues. Previous research already mentioned how communication between replication teams is important when preparing a replication package. In the end, our findings confirm problems already known by previous research.

Regarding the validation of the original paper, the execution results were converging with the baseline when running one algorithm per time. In the execution of algorithms combinations, we have seen divergent numbers between the replication and the baseline, but similar numbers across the teams. This indicates that the quality of the artifacts played a greater influence on the results than who is the researcher replicating the experiment.

Therefore, we once again emphasize the need for having clear instructions when preparing material for other researchers to use, being the most descriptive as possible. Similarly, it is essential to test all replication assets before releasing it to the public when an empirical paper is published.

BIBLIOGRAPHY

ABAL, I.; BRABRAND, C.; WASOWSKI, A. 42 variability bugs in the linux kernel: a qualitative analysis. In: *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. [S.l.: s.n.], 2014. p. 421–432.

APEL, S.; KÄSTNER, C. An overview of feature-oriented software development. *J. Object Technol.*, v. 8, n. 5, p. 49–84, 2009.

BAGHERI, E. et al. *Foreword to the special issue on empirical evidence on software product line engineering*. [S.l.]: Springer, 2016.

BALDASSARRE, M. T. et al. Replication types: Towards a shared taxonomy. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: ACM, 2014. (EASE '14), p. 18:1–18:4. ISBN 978-1-4503-2476-2.

BASILI, V. R.; SHULL, F.; LANUBILE, F. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, v. 25, n. 4, p. 456–473, Jul 1999. ISSN 0098-5589.

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

BASTOS, J. F. et al. Software product lines adoption in small organizations. *Journal of Systems and Software*, v. 131, p. 112 – 128, 2017. ISSN 0164-1212. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0164121217300997⟩.

BERGER, E. D. et al. *FSE/CACM Rebuttal²: Correcting A Large-Scale Study of Programming Languages and Code Quality in GitHub*. 2019.

Berger, T. et al. A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering*, v. 39, n. 12, p. 1611–1640, 2013.

BEZERRA, R. M. M. et al. Replication of empirical studies in software engineering: An update of a systematic mapping study. In: *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. [S.l.: s.n.], 2015. p. 1–4. ISSN 1949-3770.

BROOKS, A. et al. Replication of experimental results in software engineering. *International Software Engineering Research Network (ISERN) Technical Report ISERN-96-10, University of Strathclyde*, 1996.

CARVER, J. C. Towards reporting guidelines for experimental replications: A proposal. In: CITESEER. *1st international workshop on replication in empirical software engineering*. [S.l.], 2010. p. 2–5.

CLEMENTS, P.; NORTHROP, L. *Software product lines : practices and patterns*. Boston: Addison-Wesley, 2002. ISBN 0-201-70332-7.

CRESWELL, J. W. Research design: Qualitative, quantitative, and mixed methods approaches. SAGE Publications, 2002.

EASTERBROOK, S. et al. Selecting empirical methods for software engineering research. In: *Guide to advanced empirical software engineering*. [S.l.]: Springer, 2008. p. 285–311.

GILES, J. The trouble with replication. *Nature*, v. 442, p. 344–347, 2006.

HAN, X.; YU, T. An empirical study on performance bugs for highly configurable software systems. In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: ACM, 2016. (ESEM '16), p. 23:1–23:10. ISBN 978-1-4503-4427-2.

JURISTO, N.; MORENO, A. M. *Basics of Software Engineering Experimentation*. Boston: Kluwer Academic Publishers, 2001. ISBN 0-7923-7990-X.

JURISTO, N.; VEGAS, S. Using differences among replications of software engineering experiments to gain knowledge. In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2009. (ESEM '09), p. 356–366. ISBN 978-1-4244-4842-5.

JURISTO, N.; VEGAS, S. The role of non-exact replications in software engineering experiments. *Empirical Software Engineering*, v. 16, n. 3, p. 295–324, 2011.

KITCHENHAM, B. et al. Evaluating guidelines for empirical software engineering studies. In: *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2006. (ISESE '06), p. 38–47. ISBN 1595932186. Disponível em: ⟨https://doi.org/10.1145/1159733.1159742⟩.

KREIN, J. L.; KNUTSON, C. D. A case for replication: Synthesizing research methodologies in software engineering. In: *RESER2010: proceedings of the 1st international workshop on replication in empirical software engineering research*. [S.l.: s.n.], 2010.

LANDIS, J. R.; KOCH, G. G. An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. *Biometrics*, JSTOR, p. 363–374, 1977.

MACHADO, I. do C. et al. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology*, v. 56, n. 10, p. 1183 – 1199, 2014. ISSN 0950-5849. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0950584914000834⟩.

MAGALHAES, C. V. de et al. Investigations about replication of empirical studies in software engineering: A systematic mapping study. *Information and Software Technology*, v. 64, p. 76 – 101, 2015. ISSN 0950-5849.

MEDEIROS, F. et al. A comparison of 10 sampling algorithms for configurable systems. In: *Proceedings of the 38th International Conference on Software Engineering*. New York, NY, USA: ACM, 2016. (ICSE '16), p. 643–654. ISBN 978-1-4503-3900-1.

MENDE, T. Replication of defect prediction studies: Problems, pitfalls and recommendations. In: *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. New York, NY, USA: ACM, 2010. (PROMISE '10), p. 5:1–5:10. ISBN 978-1-4503-0404-7.

NETO, P. A. da M. S. et al. A systematic mapping study of software product lines testing. *Information and Software Technology*, v. 53, n. 5, p. 407 – 423, 2011. ISSN 0950-5849. Special Section on Best Papers from XP2010. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0950584910002193⟩.

NOLL, J.; BEECHAM, S.; SEICHTER, D. A qualitative study of open source software development: The open emr project. In: IEEE. *2011 International Symposium on Empirical Software Engineering and Measurement*. [S.l.], 2011. p. 30–39.

PERROUIN, G. et al. Automated and scalable t-wise test case generation strategies for software product lines. In: IEEE. *2010 Third international conference on software testing, verification and validation*. [S.l.], 2010. p. 459–468.

POHL, K.; BÖCKLE, G.; LINDEN, F. J. van D. *Software product line engineering : foundations, principles, and techniques*. New York, NY: Springer, 2005. ISBN 978-3-540-24372-4.

SANTOS, A. et al. A procedure and guidelines for analyzing groups of software engineering replications. *IEEE Transactions on Software Engineering*, IEEE, 2019.

SEAMAN, C. B. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, v. 25, p. 557–572, 07 1999. ISSN 0098-5589. Disponível em: ⟨doi.ieeecomputersociety.org/10.1109/32.799955⟩.

SHULL, F. et al. Replicating software engineering experiments: addressing the tacit knowledge problem. In: *Proceedings International Symposium on Empirical Software Engineering*. [S.l.: s.n.], 2002. p. 7–16.

SILVA, F. Q. B. da et al. Replication of empirical studies in software engineering research: a systematic mapping study. *Empirical Software Engineering*, Springer, v. 19, n. 3, p. 501–557, 2014. ISSN 1573-7616.

SINCERO, J. et al. Is the linux kernel a software product line. In: *Proc. SPLC Workshop on Open Source Software and Product Lines*. [S.l.: s.n.], 2007.

SOLARI, M.; VEGAS, S. Classifying and analysing replication packages for software engineering experimentation. In: *7th International Conference on Product Focused Software Process Improvement (PROFES 2006)-Workshop Series in Empirical Software Engineering (WSESE). Amsterdam, Paises Bajos*. [S.l.: s.n.], 2006.

SOLARI, M.; VEGAS, S.; JURISTO, N. Content and structure of laboratory packages for software engineering experiments. *Information and Software Technology*, v. 97, p. 64 – 79, 2018. ISSN 0950-5849. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0950584916304220⟩.

SOMMERVILLE, I. *Software engineering*. Boston: Pearson, 2011. ISBN 978-0-13-703515-1.

SORTE, M. A. L. Replication as a verification technique in survey research: A paradigm. *Sociological Quarterly*, Blackwell Publishing Ltd, v. 13, n. 2, p. 218–227, 1972. ISSN 1533-8525.

SPENCER, H.; COLLYER, G. # ifdef considered harmful, or portability experience with c news. Citeseer, 1992.

SPLC. *Call for Empirical Software Engineering Journal: special issue on "Configurable Systems"*. 2019. Disponível em: ⟨https://splc2019.net/call-for-papers/call-for-empirical-software-engineering-special-issue/⟩.

STRAUSS, A.; CORBIN, J. *Basics of qualitative research techniques*. [S.l.]: Sage publications, 1998.

WOHLIN, C. et al. *Experimentation in software engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

Appendix

A

APPENDIX A: FOCUS GROUP QUESTIONS

In the empirical study, we have structured and applied a focus group session with each group replicating the experiment. In this appendix, in Table A.1, we show the list of questions used in the sessions.

Table A.1: Questions of focus group session

Code	Question
C1	In the environment setting phase, which were the major difficulties?
C2	Some participants said the instructions were clear. Others said they here not clear. After all, in your opinion, the instructions were clear or not? Why do you think that there was this discrepancy?
C3	In your opinion, is it possible that the authors could have thought that people with a degree in computing would have at least a notion of how to install an Integrated Development Environment and that is why they did not took took up much space explaining that on the web site?
C4	Explain how you installed undertaker. Did you try to run the algorithms without undertaker installed simply to see what would happen?
E1	Was it difficult to run the algorithms? Why? List some hardships.
E2	By the forms answers, most of the people agreed that the algorithms execution tutorial (from the study's website) could be better. But do you think that a more descriptive tutorial would be only something that would be better or would it be fundamental for the replication success?
E3	During the activity I have noticed that for some teams was strange to perceive that samples/file and configurations per file were the same thing. Why do you think that, even though these concepts are defined on the paper, many people had this difficulty?
E4	Did you managed to check the algorithms' source code? Did that helped the task somehow? In what exactly?

I1	What was your overall impression about the paper (in terms of structure, writing, etc)? Do you think that the way the study was reported was appropriate for a replication? Why?
I2	In general was the paper reasonable to understand (even you not having a wide domain of its subject?)
I3	Can you recall some specific part that was not so clear?
I4	When I asked you in the form "do the results diverge", what was your comprehension about the divergence or not divergence of the results?
ER1	Was there any point on the training session you think could be better explained? How exactly? What do you think I could have done to make the concepts and the activity more clear?
ER2	Do you think my intervention was essential for you to be able to replicate the experiment? Did it help on your comprehension about the paper?
ER3	How the experiment could be better detailed in order to facilitate the replication?
ER4	Do you think this activity was useful for your academic background formation? (Please be honest; you do not have to please me at all).
ER5	Ranging from 1 to 5, what score would you give to describe how interesting the experiment was to you?
ER6	Do you think if the activity would have been applied in the beginning of the course would it change something on the experiment execution in terms of subject's motivation and time availability?
ER7	How was the task division among you (being in mind that no one will be humiliated or punished by the content of any answer given). Was it straightforward to synchronize tasks and meetings?
ER8	Do you think the lack of experience with scientific research (not restricted to Software Engineering) was something that played against you when executing the experiment's task? Why?

Appendix

B

APPENDIX B: ARTIFACTS

In this appendix, we present the artifacts produced and used for the empirical study designed. In Section B.1, there is the Consent Term in Portuguese language (all the participants spoke Portuguese as their primary language). The extraction spreadsheet used for collecting the algorithms executions comes in Section B.2. The forms used in the study are on Section B.3, namely: Participant Characterization form, Feedback form *Setting up the Environment* phase, and Post-experiment form. These artifacts can also be obtained in ⟨https://github.com/danielamador/ese_replication_assets⟩. The artifacts related to the original experiment replicated can be obtained in ⟨http://www.dsc.ufcg.edu.br/~spg/sampling/⟩.

B.1 CONSENT TERM (IN PORTUGUESE)

Abaixo seguem os termos de consentimento de participação do estudo. Ao clicar em "Próximo/Next" você estará sinalizando concordância com os termos descritos:

1. Caracterização: O estudo empírico será conduzido por Daniel Amador dos Santos, mestrando sob orientação de Eduardo Santana de Almeida, como parte do estudo "Replicating Experiments in Software Product Lines: An Empirical Study".

   Você está sendo convidado(a) a participar deste estudo empírico, que tem como finalidade investigar o processo de replicação em um experimento de engenharia de software.

2. Atividades: Estão previstas as seguintes atividades:

   (a) Preenchimento de formulários de caracterização e feedback das atividades desempenhadas.

   (b) Leitura e discussão de artigo.

   (c) Instalação de ambientes de desenvolvimento.

(d) Execução de algoritmos.

(e) Participação em Focus Group.

3. Sigilo: Em eventuais publicações científicas decorrentes do estudo atual, será preservado o anonimato do(a) participante, bem como quaisquer informações que possam identificá-lo(a).

4. Esclarecimento: O(A) participante pode requisitar a qualquer momento as finalidades do estudo, bem como seu papel em cada atividade.

5. Contato do Pesquisador (email): danielsegundoemail@gmail.com

6. Distribuição prevista das atividades:

   (a) Semana 1:

      i. Preencher Formulário 1.

      ii. Ler paper.

   (b) Semana 2:

      i. Preencher Formulários 2 e 3.

      ii. Rodar algoritmos.

      iii. Preencher planilhas de extração.

   (c) Semana 3 (Um dia apenas. A ser negociado):

      i. Sessão de Focus Group.

B.2 EXTRACTION SPREADSHEET

Team:

	1 file per time		**Restrições**		**Global**		**Headers**		**Build system**	
Algoritmo	Bugs	Samples per File	Bugs	Samples per File	Bugs	Samples per File	Bugs	Samples per File	Bugs	Samples per File
Statement-coverage										
Most-enabled-disabled										
One-enabled										
One-disabled										
Random										
Pair-wise										
Three-wise										
Four-wise										
Five-wise										
Six-wise										
C1 Pair-wise e one-disabled										
C2 One-enabled, one-disabled e statement-coverage										
C3 One-enabled, one-disabled e most-enabled-disabled										
C4 One-enabled, one-disabled e pair-wise										

B.3 FORMS

Participant Characterization

* Required

Who are you?

1. Name *

2. Age:

3. Instruction: *

   Mark only one oval.

   - ◯ Special student/Undergraduate student
   - ◯ Special student/Graduate student
   - ◯ Masters
   - ◯ Ph.D
   - ◯ Post-Doc

4. Graduation Course *

5. Have you ever participated in a controlled experiment before? *

Mark only one oval.

- ◯ No
- ◯ Yes, as a participant (subject)
- ◯ Yes, as a researcher

6. How many experiments have you applied or designed before? *

Mark only one oval.

- ◯ I didn't answer "Yes, as a researcher" on the previous question
- ◯ 1 experiment
- ◯ 2 experiments
- ◯ 3 to 5 experiments
- ◯ Above 5 experiments

7. Experience on the Market *

In activities related to Software Engineering (ex.: programming, tests, etc.)

Mark only one oval per row.

	No Experience	Below 1 year	Between 1 year (inclusive) e 2 years (exclusive)	Between 2 years (inclusive) e 5 years (exclusive)	5 years or more
Experience in years	◯	◯	◯	◯	◯

8. Knowledge about the following technologies *

Mark only one oval per row.

	I've had never heard about (before training session)	I've heard about, but I didn't have knowledge about (before training session)	I have basic notions	I have theorical knowledge	Theorical and practical experiencence
Configurable Systems	◯	◯	◯	◯	◯
Preprocessor Directives in C/C++	◯	◯	◯	◯	◯
Algorithm Complexity	◯	◯	◯	◯	◯

9. Experience in programming languages: *

Mark only one oval per row.

	No Experience	Below 1 year	Between 1 year (inclusive) e 2 years (exclusive)	Between 2 years (inclusive) e 5 years (exclusive)	5 years or more
Computer Programming (Any language)	◯	◯	◯	◯	◯
C	◯	◯	◯	◯	◯
Java	◯	◯	◯	◯	◯

10. Skills in Linux usage *

Mark only one oval.

- ◯ I've never used Linux before
- ◯ I use Linux from the GUI with difficulties
- ◯ I use Linux from the GUI without difficulties but I don't use the CLI
- ◯ I use Linux from the GUI without difficulties and I have a basic domain of the CLI
- ◯ I use Linux from the GUI without difficulties and I have a good domain of the CLI

Feedback Setting up the Environment

* Required

1. **Name:** *

2. **1 - List the difficulties you experienced while setting up the environment.** *

3. **2a - Was it possible to understand clearly what was needed to set up the environment from the paper and experiment website?** *
   Mark only one oval.
   - ◯ Yes
   - ◯ No

4. **2b - Why?** *

5. **2c - What exactly was not that clear, in case the answer for the question 2a is negative.** *

6. **3 - Did you find broken links. If so, from which artifacts?** *

7. **4a - So far, do you think that the replication artifacts were satisfactory?** *
Mark only one oval.

- ◯ Yes, they were perfectly satisfactory.
- ◯ Yes, with regards.
- ◯ No

8. **4b - Justify** *

Post-experiment

* Required

1. Group: *

Feedback of Algorithms Execution

2. Please mention difficulties regarding algorithms execution after setting up the environment, in case you have experienced any. *

3. Taking the definitions presented on the paper, was the algorithm output information clear? Why? *

4. Represented an obstacle in understanding the paper: *

Mark only one oval per row.

	I Fully agree	I partially agree	I disagree
Lack of domain in English language	◯	◯	◯
Lack of familiarity in SPL/Configurable Systems	◯	◯	◯
Explanation of basic concepts of the paper	◯	◯	◯

5. From your point of view, what was the most laborious phase: *

Mark only one oval.

- ◯ Reading and understanding the paper
- ◯ Setting up the environment
- ◯ Execution and understanding the algorithms results

6. What do you think the original authors could have done in order to make the replication easier or more clear? *

_____

_____

_____

_____

_____

Interpretation of the Results

7. Are the experiment results aligned with the original experiment or do they diverge? In case they have diverged, why do you think this divergence happened (assuming the execution environment is already set)? *

8. Based on the results you obtained, which algorithm would you recommend for a case in which someone intends to use the minimum amount of RAM space.? Why? *

Appendix

C

APPENDIX C: FULL LIST OF CANDIDATE PAPERS

In Table C.1, we present the list of candidate papers after inspecting the venues' proceedings. The following column headers were synthesized to save space width. Therefore, we present each abbreviation or acronym in boldface letters, with its full header title in the parenthesis: **HCS** (is this paper addressing highly-configurable systems?), **Exp**. (is it an experiment?), **Pack.** (Does the paper contain a replication package or a link to the artifacts used?), and **H.S.** (Does the experiment use human subjects?).

Table C.1: Full List of Candidate Papers

Venue	Paper	HCS	Exp.	Pack.	H.S.
ASE	Self-repair Through Reconfiguration: A Requirements Engineering Approach	Yes	Yes	No	No
	Flexible and Scalable Consistency Checking on Product Line Variability Models	Yes	No	-	-
	Automated Variability Analysis and Testing of an E-Commerce Site. An Experience Report	Yes	No	-	-
	Visualization-based Analysis of Quality for Large-scale Software Systems	No	-	-	-
	Model-Driven Derivation of Product Architectures	Yes	No	-	-
ESE	Do background colors improve program comprehension in the #ifdef hell?	Yes	Yes	Yes	70
	Comprehensibility of UML-based software product line specifications	Yes	Yes	Partial	116
	The effects of visualization and interaction techniques on feature model configuration	Yes	Yes	Partial	20

Table C.1 continued from previous page

ESEC FSE	Can Developer-Module Networks Predict Failures?	No	-	-	-
	Utilizing Recommender Systems to Support Software Requirements Elicitation	Yes	No	-	-
	An Empirical Comparison between Direct and Indirect Test Result Checking Approaches	No	-	-	-
	Experience Report on Software Product Line Evolution due to Market Reposition	Yes	No	Yes	Yes
	Memories of Bug Fixes	No	-	-	-
	Which Warnings Should I Fix First?	No	-	-	-
	Scalable Analysis of Variable Software	Yes	Yes	Yes	No
	Cross-project Defect Prediction	No	-	-	-
	Counterexample Guided Abstraction Refinement of Product-Line Behavioural Models	Yes	Yes	Yes	No
	A Discrete-Time Feedback Controller-for Containerized Cloud Applications	No	-	-	-
	Performance-Influence Models for Highly Configurable Systems	Yes	Yes	Yes	No
	Industrial Experience with Building a Web Portal Product Line using a Lightweight, Reactive Approach	Yes	No	-	-
	Fundamental Concepts for Practical Software Architecture	No	-	-	-
	Improving Trace Accuracy through Data-Driven Configuration and Composition of Tracing Features	Yes	Yes	No	No
	Training on Errors Experiment to Detect Fault-ProneSoftware Modules by Spam Filter	Yes	No	-	-
GPCE	Does the Discipline of Preprocessor Annotations Matter? A Controlled Experiment	Yes	Yes	Yes	18
	A Comparison of Product-based, Feature-based, and Family-based Type Checking	Yes	Yes	Yes	No
	On the Value of Learning From Defect Dense Components for Software Defect Prediction	No	-	-	-

Table C.1 continued from previous page

Do External Feedback Loops Improve the Design of Self-Adaptive Systems? A Controlled Experiment	Yes	Yes	Yes	24
Designing Search Based Adaptive Systems: A Quantitative Approach	Yes	Yes	No	No
FAVE - Factor Analysis Based Approach for Detecting Product Line Variability from Change History	Yes	Yes	No	No
An Evaluation of Argument Patterns to Reduce Pitfalls of Applying Assurance Case	No	-	-	-
Evolving an Adaptive Industrial Software System to Use Architecture-Based Self-Adaptation	Yes	Yes	No	No
The Effect of Code Coverage on Fault Detection under Different Testing Profiles	No	-	-	-
Towards More Reliable Configurators: A Re-engineering Perspective	No	-	-	-
Software Product Line Evolution: The Selecta System	Yes	No	-	-
On the Relationship between Functional Size and Software	No	-	-	-
A Tale of Migration to Cloud Computing for Sharing Experiences and Observations	No	-	-	-
QoS-Aware Fully Decentralized Service Assembly	No	-	-	-
Coordination of Distributed Systems through Self-Organizing Group Topologies	No	-	-	-
An Extension of Fault-Prone Filtering Using Precise Training and a Dynamic Threshold	No	-	-	-
Engineering Adaptation with Zanshin: An Experience Report	No	-	-	-
Efficient Runtime Quantitative Verification using Caching, Lookahead, and Nearly-Optimal Reconfiguration	Yes	Yes	Yes	No
Lifting Model Transformations to Product Lines	Yes	Yes	Yes	No

Table C.1 continued from previous page

	Can Software Engineering Students Program Defect-free? An Educational Approach	Yes	Yes	No	70
	Data Flow Testing of Service-Oriented Workflow Applications	No	-	-	-
	Experiments on Quality Evaluation of Embedded Software in Japan Robot Software Design Contest	No	-	-	-
	On Architectural Diversity of Dynamic Adaptive	Yes	Yes	No	No
	Research Journey Towards Industrial Application of Reuse Technique	Yes	Yes	No	No
	A Comparison of 10 Sampling Algorithms for Configurable Systems	Yes	Yes	Yes	No
	How Does the Degree of Variability Affect Bug Finding?	Yes	Yes	Yes	70
JSS	Dynamic adaptation of service compositions with variability models	Yes	Yes	No	No
	A method to optimize the scope of a software product platform based on end-user features	Yes	Yes	No	No
	From integration to composition: On the impact of software product lines, global development and ecosystems	Yes	No	-	-
	A feature-driven crossover operator for multi-objective and evolutionary optimization of product line architectures	Yes	Yes	No	No
	ScapeGoat: Spotting abnormal resource usage in component-based reconfigurable software systems	Yes	No	-	-
	A genetic algorithm for optimized feature selection with resource constraints in software product lines	Yes	Yes	No	No
	Handling variant requirements in domain modeling	Yes	No	-	-
	DRAMA: A framework for domain requirements analysis and modeling architectures in software product lines	Yes	No	-	-
	An approach for optimized feature selection in large-scale software product lines	Yes	Yes	Partial	No

Table C.1 continued from previous page

	An assessment of search-based techniques for reverse engineering feature models	Yes	Yes	Yes	No
	Analyzing inconsistencies in software product lines using an ontological rule-based approach	Yes	Yes	Partial	No
	Automated extraction of product comparison matrices from informal product descriptions	Yes	Yes	Partial	No
	Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications	Yes	Yes	Yes	No
	Improving feature location using structural similarity and iterative graph mapping	Yes	Yes	No	No
	Model-driven support for product line evolution on feature level	Yes	Yes	No	No
	A mixed-method approach for the empirical evaluation of the issue-based variability modeling	Yes	Yes	No	258
	Automated diagnosis of feature model configurations	Yes	Yes	No	No
	Evolving feature model configurations in software product lines	Yes	Yes	No	No
SPLC	Product-Line Maintenance with Emergent Contract Interfaces	Yes	Yes	Yes	No