Information and Software Technology 147 (2022) 106870

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Investigating replication challenges through multiple replications of an experiment

Daniel Amador dos Santos [a,*], Eduardo Santana de Almeida [a], Iftekhar Ahmed [b]

[a] *Institute of Computing, Federal University of Bahia (IC-UFBA) Salvador, BA, Brazil*
[b] *Department of Informatics, University of California, Irvine, CA, United States of America*

ARTICLE INFO

Keywords:
Replication
Empirical software engineering
Experiments
Highly-configurable systems

ABSTRACT

Context: 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.
Objective: In this study, we aim to fill this gap by investigating difficulties emerging when replicating an experiment.
Method: We used participants with distinct backgrounds to play the role of a research group attempting to replicate an experimental study addressing Highly-Configurable Systems. Seven external close replications in total were performed. After obtaining the quantitative replication results, a focus group session was applied to each group inquiring about the replication experience. We used the grounded theory's constant comparison method for the qualitative analysis.
Results: We have seen in our study that, in the replications performed, most results hold when comparing them with the baseline. However, the participants reported many difficulties in replicating the original study, mostly related to the lack of clarity of the instructions and the presence of defects on replication artifacts. Based on our findings, we provide recommendations that can help mitigate the problems reported.
Conclusions: The quality of replication artifacts and the lack of clear instructions might impact an experiment replication. We advocate having good quality replication instructions and well-prepared laboratory packages to foster and enable researchers to perform better replications.

1. Introduction

For a piece of knowledge to be considered valid, a phenomenon must be replicable and observable under different contexts [1]. Over the years, many guidelines, frameworks, and techniques have been developed to guide researchers to perform replications in Software Engineering [2–7]. Furthermore, to a limited extent, experiment papers have been published providing their assets for replication in the form of a replication package[1] [8,9].

Sometimes, a replication might refute an existing study. The authors of the original paper might disagree with the refutation, refining their arguments and reaffirming the findings of their study [10]. 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.

Overall, there is an agreement in the research community that the lack of guidelines and the non-availability of replication packages are some of the causes there are relatively few replications in Software Engineering [11]. However, when these obstacles are removed, many other challenges are still present, such as: If a researcher selects an experiment paper to replicate, does that mean they are able to execute that replication seamlessly? If not, what problems and difficulties does the researcher face when conducting the replication? Is it possible to obtain the same results as the original paper?

To fulfill the mentioned gaps and answer the questions [12,13], we conducted an empirical study in which the participants play the role of researchers conducting a software-based experiment replication. In other words, the participants are not subjects of the replication itself but rather have the task of conducting the experiment and observing the results.

We chose an experiment to replicate that investigated sampling algorithms' performance for detecting variability bugs [14]. Next, based on the focus groups transcripts, we conducted a qualitative analysis

* Corresponding author.
E-mail addresses: daniel.amador@email.com (D.A. dos Santos), esa@dcc.ufba.br (E.S. de Almeida), iftekha@uci.edu (I. Ahmed).

[1] Example of a replication package [42]: https://zenodo.org/record/4559141#.YciX_GDMJPY.

https://doi.org/10.1016/j.infsof.2022.106870
Received 13 February 2021; Received in revised form 5 January 2022; Accepted 4 February 2022
Available online 18 February 2022

using the constant comparison method [15] and identified a set of replication problems. We also provide a set of recommendations for practitioners, researchers, and educators. We conclude that the quality of the replication assets was most influential on the results of the close replication proposed.

In summary, this paper makes the following contributions:

1. Present the results of seven replications of the study *A Comparison of 10 Sampling Algorithms for Configurable Systems* [14]. We aimed at verifying if the original study results hold under a group of external replications.
2. Document challenges and difficulties when replicating an experiment in Software Engineering, taking into account the diverse expertise of the participants. Based on the evidence found, we provide recommendations for researchers, educators, and practitioners.

The remainder of this paper is structured as follows: Section 2 discusses the background and related work. In Section 3, we describe the baseline paper's design and results. Section 4 describes the methodology of the empirical study, which comprises the replications and the qualitative analysis design. The study execution is described in Section 5. Section 6 contains results and discussions. Section 7 presents recommendations for researchers, practitioners, and educators. Section 8 describes the threats to validity. Lastly, Section 9 presents the conclusions and future research directions.

2. Background and related work

In this section, we present the definitions of replication, different types of replication, and the current state of replication in Software Engineering. We also present related work.

2.1. Definitions and purpose of replication

La Sorte [16] defines that “replication refers to a conscious and systematic repeat of the original study". According to Juristo and Vegas [7], replication is “the repetition of an experiment to double-check its results". 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 work [2], also called original or baseline study.

Hardly ever, findings from a single study can be generalized to all possible contexts. Study subjects and the environment might impact the results even when the methodology is strictly designed and followed [17]. In other cases, there might be unintended bias of the research group [18]. Therefore, empirical studies should be replicated in order to make evidence valid outside the context of the original study.

To replicate studies it is necessary to transmit knowledge from the original group to the replicating one. A useful tool for that is the laboratory package. Laboratory packages or replication packages are the packed set of information and materials to use in a replication [19]. Ideally, they should contain every artifact used in the original empirical study so while 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 [20].

2.2. Replication types terminology

In the Software Engineering literature regarding replication, classifications have been developed considering the following aspects: how much involvement the baseline study's researchers keep with the replication, and how much the replication study's design is faithful to the baseline experiment. de Magalhaes et al. [2] point out those classifications naming is not consistent throughout the papers. That is why a replication needs to address and explain the terms used for defining its design.

In terms of involvement of the original researchers, we will use Brooks' definitions [3]. They define internal replication as the one performed at least by one of the researchers from the original study. Concerning the similarity of baseline and replications, we will be using classifications from Baldassarre et al. [21]. In their classification, a close replication is the one that is as similar as the original study. Therefore, for our study, we define the replication designed as being close and external.

2.3. Current state of replications in software engineering

Previously, da Silva et al. [13], de Magalhaes et al. [2], and Bezerra et al. [22] performed tertiary studies about replication in Software Engineering. In general, these tertiary studies aimed to understand which topics are addressed the most in replications. Besides papers describing replications themselves, those authors have searched for papers containing definitions, frameworks, and tools to aid replications.

In these mappings, some gaps were identified that should be filled to improve replications, both in number and quality, such as: (1) The number of replications still must grow; (2) Researchers must achieve a common ground on classifications and definitions. (3) It is desirable that baseline studies and replication are not distant in time; (4) 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.

2.4. Related work

Although replication has been an active research topic in the later years, it is still rare to find papers addressing problems when performing a replication. In one interesting instance, Mende [23] reports two replications on the defect prediction model, aiming to investigate “possible problems and pitfalls that occur during replication." This author lists the difficulties found and then elaborates a set of recommendations to perform better defect prediction model replications. His findings, however, are specific for replications addressing defect prediction studies. Thus, their recommendations cannot apply for other domains in Software Engineering.

Many other papers addressing Empirical Software Engineering mention problems researchers experienced in their replications. However, apart from [23], none of the replication studies in this section had the main intention to investigate replication problems. Instead, those problems were mentioned spontaneously but not in a systematic or structured fashion. The listing of problems can be found in Table 1.

Overall, several studies report that a new context imposes design changes [24]. Those design changes sometimes make comparison with the baseline not entirely possible [25]. However, researchers might foresee difficulties imposed by the design change and modify the replication to mitigate those difficulties [26]. Another ongoing discussion in the community is how to combine results from replications. Santos et al. [27] suggest that instead of trying simply reproducing results, baseline and replications should be seen as complementary pieces in the attempt of understanding a phenomenon.

Going further on the topic of replication design changes, one difficulty that can emerge is, on a replication with human subjects, the

Table 1
Replication-related problems identified in literature.

Type	Problem	Papers
Human subjects	New context requiring experimental design changes	[24–26,30]
	Experimental design changes make results comparison unfeasible	[25]
	Replication has fewer subjects than baseline	[28]
	Participants in replication have less time than participants in the original study	[29]
	Replication performed in a different experimental setting	[30]
Databases and data repositories	Lack of full access to original data source	[31,32]
	Unavailability of data due to change in the Version Control System	[33]

research group might have fewer participants than the baseline. It is important consider that issue when performing the analysis of the results, even though mitigation actions might not be enough, making the results not comparable to the original experiment [28].

Another possible difficulty is when the participant of the replication has less time to execute the study than the participants of the baseline [29]. This problem might introduce a critical threat to validity, which cannot be overlooked.

On a different occasion, an in-class replication might be changed to a take-home activity [30]. Researchers adopting this alternative should be aware of the consequences and attempt to mitigate possible threats planted by this modification.

In replications making heavy use of databases and repositories, some works state that it might not be possible to have full access to the original data sources, which can reduce the reproducibility of the results [31,32]. Rossi et al. [33] found that the Version Control System used in the original study was changed from CVS to SVN, and as a consequence, data available only through the older version control system was not available in the replication study.

Although prior studies cite replication problems to a limited extent, no empirical software engineering studies solely focused on understanding replication problems. In this paper we aim to fill this gap.

3. Original study

In this section, we describe the original study on which the replication was performed.

3.1. Goal and research questions of original study

The chosen study for replication was: "A Comparison of 10 Sampling Algorithms for Configurable Systems" [14]. When a configurable system is developed, there is often the need to conduct testing, similar to any other software. Testing HCSs adds an extra layer of complexity since many configurations can exist [34,35]. Unique combinations of features might trigger 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 such as 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.

Upon this premise, Medeiros et al. [14] 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 to find if applying one algorithm after another produces interesting results. We will name hereafter this investigation as **Part 1** (while in Medeiros et al.'s paper was called Study 1).

The results for this investigation are summarized in Table 2. The column "Faults" corresponds to the number of faults each algorithm is able to detect, while "Sample/File" indicates the size of the sample set. This table shows also the performance for selected algorithms combinations, displayed on the last four rows.

Table 2
Results of Study 1.
Source: Medeiros et al. [14]

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

On the other hand, as Medeiros et al. state [14], 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 assume that ignoring those four factors does not impact bug detection and the size of the sample set. The original paper performs an additional investigation, taking each factor into account when calculating the performance of the sampling algorithms. This part of the investigation, which in the baseline paper is called Study 2, will be referenced onwards as **Part 2**.

Table 3 exhibits the results found under the restrictions mentioned. The empty cells show which sampling algorithms could not be executed (due to some algorithms being not able to scale under certain circumstances).

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, the algorithms were used to detected bugs on 24 C/C++ open source projects. Those systems have a known list of bugs, and the authors intended to verify how many of these known bugs the sampling algorithms are able to catch. In the second part, only two out of the 24 systems were feasible to run against the algorithms. That happened because it is necessary to have a feature model or other asset that contains information about constraints to verify the influence that configuration restrictions impose on the algorithms' performance. Only Linux and BusyBox had such information.

The authors published a replication package, containing the assets used on the experiment for replication purposes, namely the implementation of the sampling algorithms in Java, instructions on how to run that implementation, dependencies needed to execute the Java files, and a zip file containing the open-source subject systems.[2]

4. Study design

In this section, we describe the design of our study.

[2] http://www.dsc.ufcg.edu.br/~spg/sampling/.

Table 3
Results of Study 2.
Source: Medeiros et al. [14]

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	—	—	—	—	—	—	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 ⤊ and ⤋ to represent larger changes.

4.1. 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 if there are problems specific to HCS. In this study, we answer the following research questions:

RQ1 Does the experiment confirm the baseline study results?

RQ2 Which problems or difficulties the researchers might experience in a Software Engineering experiment replication?

RQ3 Are there problems specific to an HCS experiment replication?

In order to maximize the feedback, we designed an empirical study where several subjects 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 subjects are split into groups, each one independent from the other. Since we have many independent groups obtaining their own replication results, we consider that independent replications are performed.

4.2. Activities

The activities performed by the subjects can be seen in Fig. 1. First, the participants fill the Subject Characterization Form (Form 1).[3] Based on their background, we can assemble the groups (see Section 4.3). With the groups ready, the participants would then read the baseline paper.

Later, they attempt to set up the replication environment. This involves installing an Oracle's VirtualBox virtual machine and Ubuntu14.04 operating system, installing the libraries and Integrated Development Environment (IDE). This phase also involves running one sampling algorithm so that subjects can ensure that all dependencies are correctly installed. While the virtual machine with Ubuntu is provided, the libraries and IDE must be obtained by the participants. After that, the subjects answer the Feedback After Setting Up the Environment Form (Form 2) with their experience from that phase.

Then, they execute the Sampling Algorithms, filling the Extraction Spreadsheet. After those activities, they fill the Post-experiment Form (Form 3). Finally, each team participates in the focus group session.

[3] The artifacts names are clickable, pointing to the supplementary website: https://github.com/danielamador/ist_replication_assets.

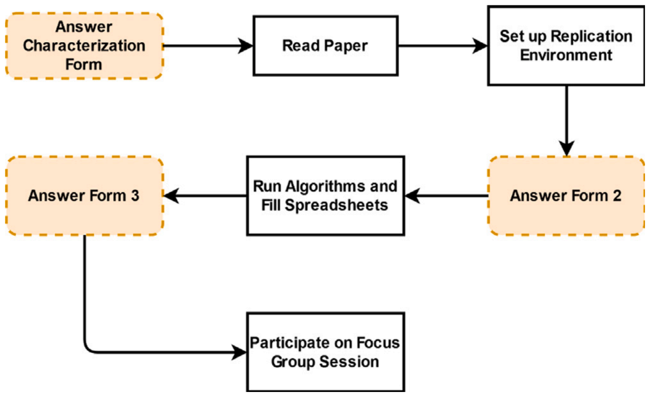
Answer Characterization Form
Read Paper
Set up Replication Environment
Answer Form 2
Run Algorithms and Fill Spreadsheets
Answer Form 3
Participate on Focus Group Session

Fig. 1. Subjects Activities.

4.3. Subjects selection process

The Subject Characterization Form was developed to collect background from the participants, and allocate them throughout the groups. This form intended to gather information, such as the subject's academic degree, knowledge about programming, experience with experiments, and knowledge about HCS. The subjects distribution, then, was designed in a manner that participants with different expertise could be placed in the groups. For instance, our idea was allocating, in the same team, a subject with reasonable experience in programming together with an experienced researcher. On the other hand, for example, another group could be formed entirely with seasoned developers. By distributing researchers with different expertise, we intend 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, Section 5.3.1), in which the most diverse variations of a phenomenon are found, the better description there will be [15]. On the other hand, it is still desirable to have also a degree of uniformity among the groups so we can capture recurring statements about a phenomenon (and then, we have a more solid grounding for that phenomenon in particular). That is why we intended to allocate experienced developers with researchers in every group.

4.4. Experiment replication phase

Before designing how the replication activity would be set up for the course, the authors tested the replication package provided on the paper website. That action was performed to guarantee that the experiment was doable from the beginning until the end (even if problems occur, but none of them preventing the replication to happen).

Table 4
Participants characterization.

Instance	Group	Name	Degree	Exp. with programming	Experience with experiments	Knowledge with HCS
1st	Group A	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 theoretical knowledge
	Group B	Saturn	Graduate	More than 5 years	Participant	Has theoretical knowledge
		Mercury	Masters Student	More than 5 years	Participant	Has theoretical knowledge
		Helios	Graduate	Between 1 and 2 years	None	Never heard before
	Group C	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 SPL
		Minerva	Masters Student	No experience	Participant	Has only basic notions about SPL
2nd	Group D	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
	Group E	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 F	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 G	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

After that, each team is given the task of replicating the experiments described by Medeiros et al. [14]. The subjects 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. The estimated time for them to execute the algorithm and extract the results is one week (counting only the execution part).

5. Study execution

Here we present the execution of the study's design outlined in the previous section.

5.1. Subjects selection

The subjects for this study were chosen by convenience. We had access to the students enrolled in a post-graduate course of Empirical Software Engineering at the Federal University of Bahia, Brazil. We invited those students to participate in the **first instance** of this study. We considered these subjects to be suitable because they were having direct contact with the experiment design in the mentioned course. Other topics related to Empirical Software Engineering were also taught, such as designing a Systematic Review and performing statistical analysis. Twelve students participated in this operation. The replication was applied as an assignment of the course, in which the participants received an extra grade to participate in the study. The professor of the discipline gently provided us three weeks to run the replication with the students. The first author also followed each team to assist the participants and verify they were performing the activity thoroughly.

In the same university, the second author of this paper leads a research laboratory focused on Software Reuse. We invited researchers from this lab to be part of the **second instance** of the experiment replication. Nine masters and Ph.D. students from this research group participated. It is important to highlight that many of these participants had designed experiments before, and we assured that each group of this instance had two participants with such expertise. Different from the first operation, this instance was not bound to any course, and the participants accepted to allocate their time to participate in the study.

The subjects characterization can be seen on Table 4. The real names were omitted for confidentiality reasons.

5.2. Focus group

After the subjects finished the study, we performed a focus group session with each team. The goal was to extend the feedback provided by them from the forms. The sessions were approximately 45-minute long. We had audio (two sources) and video recordings, according to their consent.

We prepared a set of questions to ask the subjects. However, the subjects were allowed to interact with the other ones in the focus group sessions in a manner they were able to complement each other's answers or disagree about some comment said by the others. The order of the questions could be changed if, for example, a theme addressed in a subsequent question was mentioned beforehand. After all the sessions were complete, we transcribed the focus group instances to text.

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

Environment set up refers to the phase in which the participants installed the appropriate software necessary for running the replication (as explained in Section 4.2).

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

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

Finally, we introduced questions about subjects overall impressions about participating in the empirical study. We intended to perceive how motivated subjects were and general feedback about the whole study itself. This section's answers were expected to also enrich statements coming from the previous sections, whenever possible.

5.3. Data analysis

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

5.3.1. Constant comparison method

We applied the constant comparison method for analysis. We believe this method is suitable because we primarily relied on qualitative data: the transcripts from the focus group sessions. We had other artifacts, which were also taken into consideration when applying this Grounded Theory technique, such as the explanatory notes written by the subjects on the extraction spreadsheets and the forms already mentioned.

The process, as described by Strauss and Corbin [15], is composed of Open Coding, Axial Coding and Selective Coding. Although Strauss and Corbin argue that those phases are not necessarily sequential, we describe them as if they were for simplification purposes.

According to Strauss and Corbin, coding is "the analytic process through which data are fractured, conceptualized and integrated to form theory" [15]. They also state that each code represents a phenomenon.

The open coding was performed per line. Each line might be describing phenomena besides the subject of the question. The line then is labeled according to the theme subject is referring to. A line can receive none, one or more labels, depending on how many topics the subject addresses in a single sentence.

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 [15] 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 subjects 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.

During axial coding, we started 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 subject took when facing the phenomenon, and the consequences of the phenomenon.

Next, we synthesized explanations from each coded sentence and established the links between the codes.[4]

It is important to highlight that this paper does 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 case they exist.

6. Results

We present in this section the quantitative and qualitative analysis. The quantitative analysis refers to the actual numbers obtained through the replication. The qualitative analysis is used for interpreting mainly the replication problems found and additionally other phenomena surrounding the replication.

6.1. Quantitative analysis

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

[4] The coding synthesis is available in the following address: https://github.com/danielamador/ist_replication_assets/blob/master/coding/coding_synthesis.pdf.

6.1.1. Part 1 results — ignoring limiting assumptions

This section contains the algorithms execution replication results. The results can be seen in Table 5 (for bugs) and 6 (for configuration per file). The results of the baseline study are on the first column (Baseline).

For Statement-coverage algorithm, only Groups D and E were able to obtain results. The few teams which found results got converging numbers compared to the baseline for this algorithm.

Regarding Most-enabled-disabled algorithm, all the teams were able to find the same result as the baseline. Group D 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.

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 subjects. Once again, the lack of feedback might have discouraged subjects to keep waiting until the algorithm was ran successfully. From all the teams, Groups D and E were able to execute this algorithm until the end. Both groups 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 D (Table 6) 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 E found matching results compared to the original study.

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 5 and 6. 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 E and G, which found 128. In C2, the baseline amount of bugs was 132, while the teams found 133, except group F, which found 112 bugs. C4, on the baseline contained 134 bugs, while the groups found 135 as a result, except Group F 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 B, C, and G did not deliver any results for Configurations per Sample for the algorithms combination. Groups A, D and F inserted on the table the configurations number instead (the Java implementation only gave results for bugs and configurations, but not Configuration per File). Group E attempted to calculate the results (since the total number of project files was 50 078). For instance, taking the combination C1 if 2803 (number of configurations given by the algorithms) is divided by 50 078 (total number of files) the answer is approximately 0.06, which was the number Group E 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. Part 2 results — lifting limiting assumptions

In this section, we discuss the results for the Part 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 Tables 7, 8, 9, and 10. A dash in the cell indicates that the algorithm execution is not feasible for that assumption lifted or the group left the field empty for any other reason.

Table 5
Part 1 Replication results — Bugs.

Algorithm	Base	GA	GB	GC	GD	GE	GF	GH
SC	90	—	—	—	90	90	—	—
MED	105	105	105	105	105	105	105	105
OE	107	107	107	107	107	107	107	107
OD	108	108	108	108	108	108	108	108
Random	124	—	—	—	134	124	—	—
2-W	125	125	125	125	125	125	125	125
3-W	129	129	129	129	129	129	129	129
4-W	132	132	132	132	132	132	132	132
5-W	135	135	135	135	135	135	135	135
6-W	135	135	135	135	135	135	135	135
C1	131	132	132	132	132	128	132	128
C2	132	133	133	133	133	133	112	133
C3	133	133	133	133	133	133	133	—
C4	132	135	135	135	135	135	129	135

Table 6
Study 1 Replication results — Configurations per file.

Algorithm	Base	GA	GB	GC	GD	GE	GF	GH
SC	1.3	—	—	—	1.3	1.3	—	—
MED	1.3	1.3	1.3	1.3	1.3	1.3	1.2	1.3
OE	1.7	1.7	1.7	1.7	1.7	1.7	1.7	1.7
OD	1.7	1.7	1.7	1.7	1.7	1.7	1.7	1.7
Random	2.6	—	—	—	186578	2.6	—	—
2-W	1.8	1.8	1.8	1.8	1.8	1.8	1.7	1.8
3-W	2.5	2.5	2.5	2.5	2.5	2.5	2.4	2.5
4-W	3.7	3.7	3.7	3.7	3.7	3.7	3.7	3.7
5-W	6.0	6	6	6	6	6	5.9	6
6-W	10.0	10.0	10.0	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 7
Part 2 Replication results — Constraints.

Algorithm	Base	GA	GB	GC	GD	GE	GF	GH
SC	32	32	32	—	32	32	—	32
MED	23	23	23	23	23	23	23	23
OE	30	7	7	7	7	7	7	7
OD	38	6	6	6	6	6	6	6
Random	39	—	—	—	42	44	—	—
2-W	33	—	—	—	—	—	—	—
3-W	—	—	—	—	—	—	—	—
4-W	—	—	—	—	—	—	—	—
5-W	—	—	—	—	—	—	—	—
6-W	—	—	—	—	—	—	—	—

Table 8
Part 2 Replication results — Global analysis.

Algorithm	Base	GA	GB	GC	GD	GE	GF	GH
SC	—	0	0	0	0	—	0	0
MED	27	35	35	35	35	35	35	35
OE	31	31	31	31	31	31	31	31
OD	39	38	38	38	38	38	38	38
Random	29	36	39	40	39	38	38	38
2-W	—	23	23	23	23	23	23	23
3-W	—	—	—	—	—	—	—	—
4-W	—	—	—	—	—	—	—	—
5-W	—	—	—	—	—	—	—	—
6-W	—	—	—	—	—	—	—	—

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, 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 D found 42 and Group E 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 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, 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 G 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 around 45 bugs, against 40 bugs in the original. The other algorithms matched the baseline, excluding Group G 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 G which got diverging results for all algorithms but random.

Table 9
Part 2 Replication results — Header files.

Algorithm	Base	GA	GB	GC	GD	GE	GF	GH
SC	—	0	0	0	0	31	31	0
MED	27	35	35	35	35	35	35	—
OE	31	31	31	31	31	31	31	—
OD	39	39	39	39	39	39	39	—
Random	40	47	43	44	44	42	46	45
2-W	39	39	39	39	39	39	39	39
3-W	43	45	45	45	45	45	45	45
4-W	45	46	46	46	46	46	46	46
5-W	—	—	—	—	—	—	—	—
6-W	—	—	—	—	—	—	—	—

Table 10
Part 2 Replication results — Build system.

Sampling	Base	GA	GB	GC	GD	GE	GF	GH
SC	25	31	31	31	0	31	31	0
MED	26	28	28	28	28	28	28	35
OE	20	20	20	20	20	20	20	31
OD	39	39	39	39	39	39	39	38
Random	41	42	45	40	38	41	42	38
2-W	33	33	33	36	33	33	33	33
3-W	42	46	46	46	46	46	46	—
4-W	45	45	45	45	45	45	45	—
5-W	47	47	47	47	47	47	47	—
6-W	47	47	47	47	47	47	47	—

6.1.3. *Quantitative analysis summary*

Upon the numbers found, described, and discussed in the current section, we can answer the following research question:

RQ1 Does the experiment confirm the baseline study results?

For the individual algorithms' execution ignoring the limiting assumptions, we can state that the replication results are congruent with the baseline. In almost all cases, the results found matched the original results for bugs and configurations per file. For the algorithms' combinations, the replications did not confirm the baseline for configuration per file, but the results were close for the number of bugs. In other words, we state the following observation:

> **Observation 1**: In general, the baseline results could be reproduced for individual sampling algorithms assuming that C/C++ header files, global analysis, configuration constraints and the build system do NOT interfere on results.

For the algorithms' execution lifting the assumptions, we have observed an oscillation on the numbers: many results match, and others do not. In other cases, algorithms that were not supposed to be executed can return results. However, most of the teams obtained similar results between themselves. Therefore, the results obtained in the replication cannot confirm the baseline with confidence for those cases. We synthesize the previous statement on the second observation:

> **Observation 2**: Overall, the baseline results could NOT be reproduced for individual and combined sampling algorithms assuming that C/C++ header files, global analysis, configuration constraints and the build system interfere on results.

From all the results, we could observe that the numbers found from the teams, in general, were similar. This means that, even though there was a difference in the background of the participants on the first (Groups A to D) and second instances (Groups E to G), that difference did not manifest in the replication results. However, the precise influence the background exerts on the results of experiments executed on the computer (such as the one we observed) is a matter of future research. Therefore, we synthesize the previous statement allows us to make the following observation:

> **Observation 3**: The replication package artifacts might exert a greater influence on results than the background of the participants.

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 subsection, we present the outcomes of this step.

6.2.1. *Open coding*

During the open coding, a total of 91 codes were generated. Later, the codes were grouped in higher-level categories, such as: environment set up, algorithms execution, paper interpretation, subjects replication experiment (which are the questions sections of the focus group), comments on the virtual machine, subjects' attitude, subjects' complaints, and miscellaneous (codes not related to any of the previous categories).[5] 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 [36]. 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 [37], our kappa value of 0.88 indicate almost perfect agreement among raters.

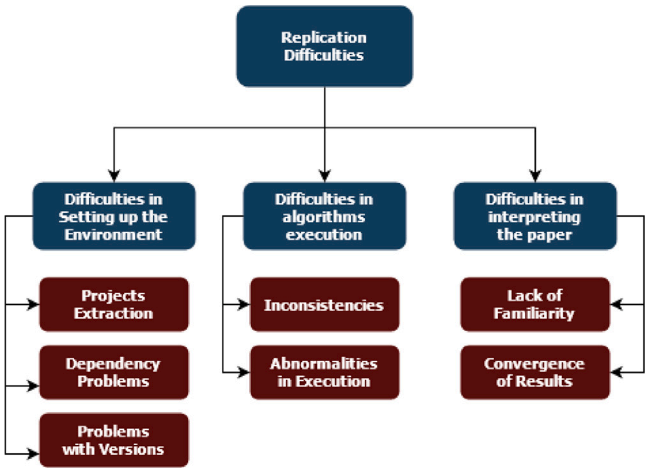
Replication Difficulties
Difficulties in Setting up the Environment
Difficulties in algorithms execution
Difficulties in interpreting the paper
Projects Extraction
Dependency Problems
Problems with Versions
Inconsistencies
Abnormalities in Execution
Lack of Familiarity
Convergence of Results

Fig. 2. Categories in Axial Coding.

6.2.2. *Axial coding*

During the Axial coding phase, we set "Replication Difficulties" as a central phenomenon (explained in Section 5.3.1). Then, we reallocated the codes under three branches. It means that for each branch, we proceed to find: the causal events, the actions subjects execute on those events, and the consequences of those actions. The initial high-level categories, showed in Fig. 2, 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). From there on, other phenomena emerged, enriching each of the categories initially defined (listed and explained in the following subsections).

6.2.3. *Setting up the environment*

Throughout the coding, ten 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 A judged this task of placing the projects folder on the appropriate location to be straightforward (Odin[6]: "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 B, 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'.")

[5] The full list of codes and their description can be found at the supplemental material: https://github.com/danielamador/ist_replication_assets/blob/master/coding/codes_list.pdf.

[6] Table 4 shows the subjects allocation. Actual names are omitted for confidentiality issues.

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 subjects figured out themselves or required assistance to the first author to overcome this difficulty.

One dependency, called “undertaker", was often cited by the subjects 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 subjects 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 subjects 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 is 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).

6.2.4. *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 inconsistencies 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 subjects which do not fit in any category listed.

Group A 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 G (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 A, B, E, and G 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. 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 subjects to wonder if the execution was running appropriately or not.

Another problem mentioned was that the algorithms execution output was not clear enough. Groups D, E,F and G mentioned issues related to execution’s output clarity. In Dionysus’s (Group E) opinion, the verbosity level was not adequate for the random algorithm (which, differently from most algorithms, showed additional information besides the execution that was not useful for a replication context). Zeus (Group G) 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.")

6.2.5. *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 subjects 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 subjects’ understanding about replications, Group B reported that they were unsure about how much intervention on the source code they were allowed to do to not invalidate the replication. One subject 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 [21]. This indicates that not being familiar with the definitions can impact the way these studies are conducted.

6.2.6. *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 are able to answer the following research questions:

RQ2 Which problems or difficulties the researchers might experience in a Software Engineering experiment replication?

We identified the following categories embracing most of the codes (Table 11): 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 subjects 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 subjects into difficulties (tagged in code lack of familiarity). This indicates that experience with replications and the area under study might contribute positively to replication success.

RQ3 Are there problems specific to an HCS experiment replication?

None of the testimonials collected indicates that any of the difficulties reported are related exclusively to HCS.

7. Recommendations

In this section we discuss our findings and based on that we highlight some recommendations for researchers, practitioners and educators.

Table 11
Codes addressing RQ2.

Category	Codes Embraced
Imprecision on Replication Instructions	Projects Extraction, Dependency Problems, Name Inconsistencies, Long Wait Time, Execution Output
Defects on the Provided Software	Dependency Problems Execution Abnormalities
Other	Lack of Familiarity

7.1. Researchers

For the researchers replicating an experiment, it is highly beneficial to communicate with the original research team [2]. This communication can minimize failures in replication results [38], but it must be done diligently since the participation of the baseline study's authors may introduce unintended bias [39]. Thus, while collaboration is essential for the progress of scientific knowledge, the independence between the original team and the researchers replicating the experiment must be preserved.

In the testimonials provided by the subjects 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 [20]. 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 [4,40] 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 [5]. We suggest using, for instance, Solari et al. proposal's to structure a replication package [19], which, on their research, have shown that using a well-structured replication package can be beneficial in this direction. Those authors also recognize that existing empirical research in Software Engineering fails to provide laboratory packages with sufficient instructions, while they point out that many other areas are more mature in that concern (as we can see, for instance, in Nature Protocol Exchange[7]). In this direction, we can infer that improving the quality instructions on replication packages in empirical software engineering helps this area to grow more mature in terms of replications.

Similarly to the approach cited before, following ACM standards[8] for artifacts can be a good reference for structuring laboratory packages.

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 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.

7.2. Practitioners

In the original paper, Medeiros et al. [14] 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 [41]. 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 latter 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).

7.3. Educators

The first operation of the exploratory study was applied with students enrolled in a course of Empirical Software Engineering. Some subjects in this round were having the first contact with experiment design. On the focus group, a subject 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. 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 subjects 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.

8. Threats to validity

We have taken care to ensure that our results are unbiased, and we have tried to eliminate the effects of random noise, but it is possible that our mitigation strategies may not have been effective. In this section, we are going to discuss threats to validity for our study.

[7] http://www.nature.com/protocolexchange/.
[8] https://www.acm.org/publications/policies/artifact-review-badging.

8.1. Internal validity

One issue that might have influenced how subjects 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 subjects were not vested as much as the researchers of the original paper. This motivation might have led the subjects 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 subjects to reason about what the algorithms' output mean, instead of only "copying and paste" the results onto the cells.

Also, as mentioned in Section 6.2.6, one of the topics mentioned in the coding process is the lack of familiarity with the themes addressed and replication itself. The population available for this study was composed mostly of novice researchers and students with no previous experience in Empirical Studies. On some occasions, even seasoned researchers with no significant experience in replications are not expert enough to conduct a replication accordingly. That represents a major threat since the replication activity usually requires expertise in this specific topic to be performed successfully.

8.2. External validity

The replication difficulties found in this paper might not extend to other replications. This experiment replication referred to the execution of many algorithms in a computer environment. Experiments with a different nature might not benefit from the findings of this study. Additionally, it is not possible to guarantee that the difficulties found by our subjects apply to a more mature and experienced set of participants.

8.3. Construct validity

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

Furthermore, since this study relies heavily on Grounded Theory's constant comparison technique, the testimonials interpretation lies 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 the interpretations.

9. Conclusion

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 Highly-Configurable Systems.

We have developed a research methodology in which the subjects emulated researchers replicating the experiment. We had in total seven 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 coding established initially four categories of challenges pertaining to replication process for the chosen experiment: Setting Up the Environment, Algorithms Execution, Paper Interpretation and Replication Experience (which intersects with all categories before). From the subjects' 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.

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 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. 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.

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.

We intend to perform also experiment replications with human subjects to investigate difficulties in this matter. We assume that empirical studies that use human subjects on their design can exhibit different challenges. As a matter of comparison, in our study, we used human subjects only in the upper empirical study, but the experiment itself was executed entirely on the computer.

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.

Additionally, since we could not detect the influence of the subject's background on the replication, there is a research opportunity for investigating that issue. Instead of balanced groups, a study can be designed to make groups with different skills perform a replication, capturing their results and the participants' perceptions.

CRediT authorship contribution statement

Daniel Amador dos Santos: Methodology, Validation, Investigation, Resources, Writing – original draft. **Eduardo Santana de Almeida:** Conceptualization, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Iftekhar Ahmed:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was partially funded by Fundação de Amparo à Pesquisa do Estado da Bahia, Brazil (FAPESB) scolarship BOL0644/2016.

References

[42] K. Revoredo, D. Djurica, J. Mendling, A study into the practice of reporting software engineering experiments, Empir. Softw. Eng. 26 (6) (2021) 1–50.

[1] N. Juristo, A.M. Moreno, Basics of Software Engineering Experimentation, Kluwer Academic Publishers, Boston, 2001.

[2] C.V. de Magalhaes, F.Q. da Silva, R.E. Santos, M. Suassuna, Investigations about replication of empirical studies in software engineering: A systematic mapping study, Inf. Softw. Technol. 64 (2015) 76–101, http://dx.doi.org/10.1016/j.infsof.2015.02.001.

[3] A. Brooks, J. Daly, J. Miller, M. Roper, M. Wood, Replication of experimental results in software engineering, in: International Software Engineering Research Network (ISERN) Technical Report ISERN-96-10, University Of Strathclyde, 1996.

[4] B. Kitchenham, H. Al-Khilidar, M.A. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, L. Zhu, Evaluating guidelines for empirical software engineering studies, in: Proceedings Of The 2006 ACM/IEEE International Symposium On Empirical Software Engineering, in: ISESE '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 38–47, http://dx.doi.org/10.1145/1159733.1159742.

[5] V.R. Basili, F. Shull, F. Lanubile, Building knowledge through families of experiments, IEEE Trans. Softw. Eng. 25 (4) (1999) 456–473.

[6] M.G. Mendonça, J.C. Maldonado, M.C. De Oliveira, J. Carver, S.C. Fabbri, F. Shull, G.H. Travassos, E.N. Höhn, V.R. Basili, A framework for software engineering experimental replications, in: 13th IEEE International Conference On Engineering Of Complex Computer Systems, ICECCS 2008, IEEE, 2008, pp. 203–212.

[7] N. Juristo, S. Vegas, 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, in: ESEM '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 356–366, http://dx.doi.org/10.1109/ESEM.2009.5314236.

[8] S. Vegas, N. Juristo, A. Moreno, M. Solari, P. Letelier, Analysis of the influence of communication between researchers on experiment replication, in: Proceedings Of The 2006 ACM/IEEE International Symposium On Empirical Software Engineering, in: ISESE '06, ACM, New York, NY, USA, 2006, pp. 28–37, http://dx.doi.org/10.1145/1159733.1159741.

[9] M. Solari, S. Vegas, 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. Amsterdam, Netherlands, WSESE, 2006.

[10] E.D. Berger, P. Maj, O. Vitek, J. Vitek, FSE/CACM rebuttal^2: Correcting a large-scale study of programming languages and code quality in GitHub, 2019, arXiv:1911.11894.

[11] M. Cruz, B. Bernardez, A. Duran, J.A. Galindo, A. Ruiz-Cortes, Replication of studies in empirical software engineering: A systematic mapping study, from 2013 to 2018, IEEE Access 8 (2019) 26773–26791.

[12] F.J. Shull, J.C. Carver, S. Vegas, N. Juristo, The role of replications in empirical software engineering, Empir. Softw. Eng. 13 (2) (2008) 211–218.

[13] F.Q.B. da Silva, M. Suassuna, A.C.C. França, A.M. Grubb, T.B. Gouveia, C.V.F. Monteiro, I.E. dos Santos, Replication of empirical studies in software engineering research: a systematic mapping study, Empir. Softw. Eng. 19 (3) (2014) 501–557, http://dx.doi.org/10.1007/s10664-012-9227-7.

[14] F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, S. Apel, A comparison of 10 sampling algorithms for configurable systems, in: Proceedings Of The 38th International Conference On Software Engineering, in: ICSE '16, ACM, New York, NY, USA, 2016, pp. 643–654, http://dx.doi.org/10.1145/2884781.2884793.

[15] A. Strauss, J. Corbin, Basics Of Qualitative Research Techniques, Sage publications, 1998.

[16] M.A. La Sorte, Replication as a verification technique in survey research: A paradigm, Sociol. Q. 13 (2) (1972) 218–227, http://dx.doi.org/10.1111/j.1533-8525.1972.tb00805.x.

[17] J. Lung, J. Aranda, S. Easterbrook, G. Wilson, On the difficulty of replicating human subjects studies in software engineering, in: 2008 ACM/IEEE 30th International Conference On Software Engineering, IEEE, 2008, pp. 191–200.

[18] J.L. Krein, C.D. Knutson, A case for replication: Synthesizing research methodologies in software engineering, in: Proceedings Of The 1st International Workshop On Replication In Empirical Software Engineering Research, in: RESER 2010, Citeseer, 2010.

[19] M. Solari, S. Vegas, N. Juristo, Content and structure of laboratory packages for software engineering experiments, Inf. Softw. Technol. 97 (2018) 64–79, http://dx.doi.org/10.1016/j.infsof.2017.12.016, URL http://www.sciencedirect.com/science/article/pii/S0950584916304220.

[20] F. Shull, V. Basili, J. Carver, J.C. Maldonado, G.H. Travassos, M. Mendonca, S. Fabbri, Replicating software engineering experiments: addressing the tacit knowledge problem, in: Proceedings International Symposium On Empirical Software Engineering, 2002, pp. 7–16, http://dx.doi.org/10.1109/ISESE.2002.1166920.

[21] M.T. Baldassarre, J. Carver, O. Dieste, N. Juristo, Replication types: Towards a shared taxonomy, in: Proceedings Of The 18th International Conference On Evaluation And Assessment In Software Engineering, in: EASE '14, ACM, New York, NY, USA, 2014, pp. 18:1–18:4, http://dx.doi.org/10.1145/2601248.2601299.

[22] R.M.M. Bezerra, F.Q.B. da Silva, A.M. Santana, C.V.C. Magalhaes, R.E.S. Santos, 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), 2015, pp. 1–4, http://dx.doi.org/10.1109/ESEM.2015.7321213.

[23] T. Mende, Replication of defect prediction studies: Problems, pitfalls and recommendations, in: Proceedings Of The 6th International Conference On Predictive Models In Software Engineering, in: PROMISE '10, ACM, New York, NY, USA, 2010, pp. 5:1–5:10, http://dx.doi.org/10.1145/1868328.1868336.

[24] A.C. Dias-Neto, G.H. Travassos, Evaluation of {model-based} testing techniques selection approaches: An external replication, in: Empirical Software Engineering And Measurement, 2009. ESEM 2009. 3rd International Symposium On, IEEE, 2009, pp. 269–278.

[25] A.M. Fernández-Sáez, M. Genero, D. Caivano, M.R. Chaudron, Does the level of detail of UML diagrams affect the maintainability of source code? A family of experiments, Empir. Softw. Eng. 21 (1) (2016) 212–259, http://dx.doi.org/10.1007/s10664-014-9354-4.

[26] L. Guerrouj, M. Di Penta, Y.-G. Guéhéneuc, G. Antoniol, An experimental investigation on the effects of context on source code identifiers splitting and expansion, Empir. Softw. Eng. 19 (2014) http://dx.doi.org/10.1007/s10664-013-9260-1.

[27] A. Santos, S. Vegas, M. Oivo, N. Juristo, Comparing the results of replications in software engineering, Empir. Softw. Eng. 26 (2021) http://dx.doi.org/10.1007/s10664-020-09907-7.

[28] C. Apa, O. Dieste, et al., Effectiveness for detecting faults within and outside the scope of testing techniques: an independent replication, Empir. Softw. Eng. 19 (2) (2014) 378–417.

[29] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, E. Astesiano, On the effectiveness of screen mockups in requirements engineering: results from an internal replication, in: Proceedings Of The 2010 ACM-IEEE International Symposium On Empirical Software Engineering And Measurement, ACM, 2010, p. 17.

[30] M. Riaz, J. King, J. Slankas, L. Williams, F. Massacci, C. Quesada-López, M. Jenkins, Identifying the implied: Findings from three differentiated replications on the use of security requirements templates, Empir. Softw. Eng. 22 (4) (2017) 2127–2178.

[31] F. Calefato, D. Gendarmi, F. Lanubile, Investigating the use of tags in collaborative development environments: A replicated study, in: Proceedings Of The 2010 ACM-IEEE International Symposium On Empirical Software Engineering And Measurement, in: ESEM '10, ACM, New York, NY, USA, 2010, pp. 24:1–24:9, http://dx.doi.org/10.1145/1852786.1852818.

[32] M. Caneill, S. Zacchiroli, Debsources: Live and historical views on macro-level software evolution, in: Proceedings Of The 8th ACM/IEEE International Symposium On Empirical Software Engineering And Measurement, 2014, pp. 1–10.

[33] B. Rossi, B. Russo, Evolution of design patterns: A replication study, in: Proceedings Of The 8th ACM/IEEE International Symposium On Empirical Software Engineering And Measurement, in: ESEM '14, ACM, New York, NY, USA, 2014, pp. 38:1–38:4, http://dx.doi.org/10.1145/2652524.2652563.

[34] P.A. da Mota Silveira Neto, I. do Carmo Machado, J.D. McGregor, E.S. de Almeida, S.R. de Lemos Meira, A systematic mapping study of software product lines testing, Inf. Softw. Technol. 53 (5) (2011) 407–423, http://dx.doi.org/10.1016/j.infsof.2010.12.003, URL http://www.sciencedirect.com/science/article/pii/S0950584910002193, Special Section on Best Papers from XP2010.

[35] I. do Carmo Machado, J.D. McGregor, Y.C. Cavalcanti, E.S. de Almeida, On strategies for testing software product lines: A systematic literature review, Inf. Softw. Technol. 56 (10) (2014) 1183–1199, http://dx.doi.org/10.1016/j.infsof.2014.04.002, URL http://www.sciencedirect.com/science/article/pii/S0950584914000834.

[36] J. Noll, S. Beecham, D. Seichter, A qualitative study of open source software development: The open EMR project, in: 2011 International Symposium On Empirical Software Engineering And Measurement, IEEE, 2011, pp. 30–39.

[37] J.R. Landis, G.G. Koch, An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers, Biometrics (1977) 363–374.

[38] J. Giles, The trouble with replication, Nature 442 (2006) 344–347.

[39] B. Kitchenham, The role of replications in empirical software engineering–a word of warning, Empirical Softw. Engg. 13 (2) (2008) 219–221, http://dx.doi.org/10.1007/s10664-008-9061-0.

[40] J.C. Carver, Towards reporting guidelines for experimental replications: A proposal, in: 1st International Workshop On Replication In Empirical Software Engineering, Citeseer, 2010, pp. 2–5.

[41] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.