EasyChair Preprint

№ 10357

Do Developers Benefit from Recommendations When Repairing Inconsistent Design Models? a Controlled Experiment

Luciano Marchezan, Wesley K. G. Assunção,
Gabriela Karoline Michelon and Alexander Egyed

June 7, 2023

Do Developers Benefit from Recommendations when Repairing Inconsistent Design Models? a Controlled Experiment

Luciano Marchezan
Wesley K. G. Assunção
ISSE - Johannes Kepler University Linz
Linz, Austria

Gabriela K. Michelon
Alexander Egyed
ISSE - Johannes Kepler University Linz
Linz, Austria

ABSTRACT

Repairing design models is a laborious task that requires a considerable amount of time and effort from developers. Repair recommendation (RR) approaches focus on reducing the effort and improving the quality of the repairs performed. Such approaches have been evaluated in terms of scalability, correctness, and minimalism. These evaluations, however, have not investigated how developers can benefit from using RRs and how they perceive the difficulty of applying RRs. Investigating and discussing the use of RRs from the developers' perspective is important to demonstrate the benefits of applying such approaches in practice. We explore this opportunity by conducting a controlled experiment carried out with 24 developers where they repaired UML design models in eight different tasks, with and without RRs. The findings indicate that developers can benefit from RRs in complex tasks by improving their effectiveness and efficiency. The results also evidence that the use of RRs does not impact the developers' perceived difficulty and confidence when repairing models. Furthermore, our findings show that not all developers choose the same RR, but rather, have varied preferences. Thus, the provision of RRs leads to developers considering additional alternatives to repair an inconsistency.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; • **Software and its engineering** → *Model-driven software engineering*.

KEYWORDS

Controlled Experiment, Consistency Checking, Repair Recommendations

1 INTRODUCTION

Inconsistent models need to be repaired to maintain their correctness and reduce the chance of requirements not being met [13, 25]. Repairing models, however, is an error-prone and laborious task that requires a considerable amount of time and effort from developers [22]. Hence, repair recommendation (RR) approaches can be applied to maintain consistency and reduce the time and effort required for the repair task, mainly on UML models [2].

Approaches proposing the use of RRs have been evaluated in a variety of scenarios in terms of scalability, correctness, and minimalism [1, 16, 20]. However, there are still limitations in this regard. Firstly, most studies in the RR field do not focus their evaluations on the perspective of developers regarding the provision of RRs *(Limitation 1)*. These evaluations are important to analyze the implications of applying RRs strategies from the perspective of its end-users, i.e., developers [19]. This is more important considering that RRs approaches are not yet adopted in industrial settings, but rather, developers perform manual repairs on demanded [10]. One reason for that may be the lack of empirical studies with human participants measuring the benefits and drawbacks of applying RRs. Secondly, as developers have different experiences and preferences, ranking or automatically executing RRs to fix inconsistencies may change the models in a way not desired by the developer (*Limitation 2*). Hence, it is important to understand the developers' preferences when repairing models before applying these approaches in industrial settings.

We address the aforementioned limitations by conducting an experiment with developers. In this paper, we report this experiment that is guided by three research questions (RQ): **RQ1**. Do developers benefit from recommendations when repairing inconsistent design models? **RQ2**. How do developers perceive the use of repair recommendations when repairing inconsistent design models? **RQ3**. Do developers have preferred recommendations when repairing inconsistent design models?

The data used to answer the RQs was collected by analyzing developers repairing inconsistent models with and without the provision of RRs. The sample was composed of 24 M.Sc./Ph.D. students with varied software development experience, ranging from academic only to more than five years of experience in the industry. To answer RQ1, we measure how the use of RRs can bring benefits to developers in terms of effectiveness (i.e., inconsistencies fixed) and efficiency (i.e., the time required) when repairing design models compared to when RRs are not provided. For RQ2, we asked developers to give us feedback regarding the difficulty and their confidence when repairing the models. To obtain the results of RQ3, we ask the developers to select and rank RRs and create new ones to understand their preferences for different tasks.

Results show that the provision of RRs benefits developers by improving their effectiveness by 37.63% (p-value of 0.04) and efficiency by 29.81% (p-value of 0.17) in comparison to not having RRs (RQ1). In more simple tasks, however, RRs reduce the efficiency of developers by increasing the time required to repair the inconsistency. These findings evidence that RRs approaches can bring benefits to developers in more complex tasks. Furthermore, the perceived difficulty and confidence of developers when RRs are given are similar without RRs (RQ2). This result demonstrates that providing RRs does not affect the developers' perspective during the repairing process. We also observed that developers do not have an ideal RR for a given inconsistency, but rather have different preferences regarding how to repair a model (RQ3). Moreover, in some contexts, any RR may be considered not applicable. These findings highlight the importance of having developers' feedback when repairing models since they have different preferences. Such

results also indicate that applying automatic RRs in models by selecting the "most suited" RR may not be ideal, as developers do not have the same opinion about what is the "most suited" RR.

This work is organized as follows: Section 2 describes the background of repair generation as well as related work. Section 3 presents the design and the threats to the validity of the controlled experiment. Results and discussion of the RQs are presented in Section 4. Lastly, Section 5 presents our conclusions.

2 BACKGROUND AND RELATED WORK

In this section, we present the definitions related to the generation of repair recommendations as well as related work.

2.1 Repair Recommendations

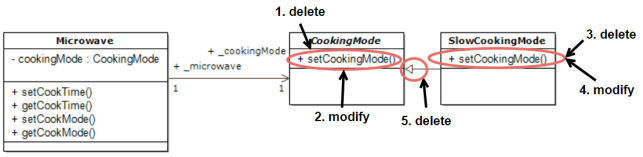
Repair recommendations are applied in models to repair inconsistencies. In this study, we consider a model as consisting of model elements that contain properties, e.g., a UML model. An example is illustrated in Figure 1, where the model contains a model element called *CookingMode* of the type class, which contains properties such as *setCookingMode* of the type operation.

Consistency Rules (CR) are applied to identify inconsistencies in the models. A CR is a condition that evaluates to a Boolean value as *true* (consistent) or *false* (inconsistent). A CR is defined for a context, which is a type of model element, e.g., UML class. Figure 1 presents the application of a CR into a model, checking if a class contains operations with the same signature. An inconsistency is found as the class *SlowCookingMode* has two operations *setCookingMode* with the same signature, one from itself and the other from the superclass *CookingMode*.

Once inconsistencies are found, repair recommendations (RR) are generated to fix them. An RR identifies the operator, the model element, the model element property, and, optionally, a value to change the model element property to, resolving an inconsistency. The following operators are possible: *add* a model element to the model or to a collection of model elements, *delete* a model element from the model or from a collection of model elements, and *modify* a model element property. Figure 1 presents examples of five RRs for the inconsistency found in the class *SlowCookingMode*. The first RR (*1. Delete*) is a concrete RR that recommends the deletion of the operation *setCookingMode* from the *CookingMode* superclass. In this case, the model element is the *CookingMode* class, the property is the list of operations, the value is the operation *setCookingMode*, and the operator is *delete*. A model-based representation of this RR would be: $(delete, CookingMode.operations, setCookingMode)$.

The generation of RRs used in this study is an adapted version of approaches found in the literature [12, 16, 22], One important adaptation is transforming the way RRs are displayed. Instead of using a model-based representation, we represent RRs using natural language (as shown in Figure 1). For instance, we transformed the aforementioned RR into "*Delete* operation *setCookingMode* from *CookingMode*" as displayed in the RR *1.Delete* in Figure 1. Also, there are different rationales for selecting approaches already present in the literature [12, 16, 22] as the basis for our RR generation. For instance, their robustness, since they have been evaluated in large-scale models in terms of scalability, correctness, and minimalism. Furthermore, the documentation provided by them was used to implement the approach using their algorithm logic and repair generator functions.[1]

CR	Class' Operations must not have the same signature.
Inconsistency	**SlowCookingMode** has two operations *setCookingMode* with the same signature (one from itself and the other from its superclass **CookingMode**)



Rank	RR Id	Repair Recommendation
	1.Delete	Delete operation **setCookingMode()** from **CookingMode**
	2.Modify	Modify the signature of operation **setCookingMode()** from **CookingMode**
	3.Delete	Delete operation **setCookingMode()** from **SlowCookingMode**
	4.Modify	Modify the signature of operation **setCookingMode()** from **SlowCookingMode**
	5.Delete	Delete inheritance association between **SlowCookingMode** and **CookingMode**

Figure 1: Example of task with Repair Recommendations

One example of an RR generation function can consider Figure 1, where the defined CR iterates over all operations of a class. The operations from the class and superclasses are compared considering their signature, i.e., name and parameters. If at least one comparison evaluates to *false*, i.e., two operations have the same signature, the class is considered inconsistent. The operations that resulted in *false* are returned in a set of model elements that are the *cause* of the inconsistency. Considering the example in Figure 1, one inconsistency is identified for class *SlowCookingMode*. This inconsistency is related to three model elements (highlighted with an ellipse), namely operation *setCookingMode* from *CookingMode*, the inheritance association from *SlowCookingMode* to *CookingMode*, and the operation *setCookingMode* from *SlowCookingMode*. These three model elements are the *cause* of the inconsistency and should be modified to repair the model.

The RR approach applies generator functions to the cause of the inconsistency to generate RRs. In this case, the generator function is applied to a universal quantifier (*forAll*) as described by the OCL[2] definition of the CR.[1] This function suggests both modifying or deleting the inconsistent model elements [16, 22]. Additionally, it suggests deleting the inheritance association, since this association is responsible for the set of operations. This leads to five RRs being generated, as shown in Figure 1. Thus, for each type of OCL expression, e.g., *exists*, *and*, *or*, among others, a generator function is applied to generate the RRs. Only one of the generated RRs is required to fix any given inconsistency. The approach we adopt treats every option equally, thus, they are not ranked in any particular order. Hence, the rank column remains empty, and it is one of the goals of the participants from the experiment.

[1]Details about the repair generation as well as all artifacts of the experiment are available at our online appendix [15]
[2]Object Constraint Language: https://www.omg.org/spec/OCL/

2.2 Related Work

Several approaches proposed the use of RRs [13]. These approaches use different strategies for generating repairs, such as relying on OCL rules and generator functions [9, 12, 16, 22], analyzing the history of changes [14, 20], and applying learning algorithms [1]. These approaches have been evaluated for their correctness, scalability, and minimalism. To the best of our knowledge, however, the use of repair recommendations has never been evaluated from the developers' perspective in a controlled experiment. Moreover, there is a lack of studies analyzing the implications of providing RRs to developers, such as benefits and preferences. Analyzing these implications is important to understand why, despite the importance of RRs, they are still not being supported by tools that are adopted in industrial scenarios [10]. Furthermore, the results of such a study can support the claims that *using RRs is better than doing manual repairs*, which is always assumed, although empirical evidence is not provided in the literature. Thus, we designed our experiment intending to collect and analyze these implications.

When designing our experiment, we applied the guidelines for experimentation with human participants in software engineering [11, 27]. Furthermore, we considered how other experiments with human participants were designed. Petersen et al. [21] report an experiment to analyze the impact of time-controlled reading inspection regarding effectiveness and efficiency. When formulating two metrics for RQ1 (see Section 3.1.1), namely, effectiveness and efficiency, we considered Petersen et al. [21] formulas to calculate the scores for each task of our experiment. When designing the methods for analyzing the results of our experiment, we considered studies that had similar protocols. For instance, Dzidek et al. [6] report an empirical evaluation of the costs and benefits of using UML documentation for software maintenance. Their experiment focused on how the use of UML impacts the time and quality of source code maintenance in comparison to the lack of documentation. This is similar to how we compare the benefits and perceived difficulty and confidence when repairing inconsistent models with and without RRs. Their experiment was also composed of quantitative and qualitative RQs, similar to ours, as RQ1 and RQ3 are quantitative and RQ2 is qualitative. Analyzing works that deal with questionnaires is also important when formulating the questions, and metrics for analyzing the results of questionnaires [18].

Since the experiment's sample is composed mostly of developers that are M.Sc./Ph.D. students, we have to consider guidelines regarding students as participants in experiments [7, 8]. Falessi et al. [7], for instance, proposes the R^3 scheme (Real, Relevant, and Recent) for classifying participants' experiences. Based on this scheme, every characterization question for the Real and Relevant experience has four alternatives: i) No experience; ii) Less than 2 years of experience (short); iii) between 2 and 5 years of experience (medium); iv) more than 5 years of experience (long). Furthermore, questions related to Recent experience had the following alternatives: i) None; ii) more than 5 years ago (old); iii) between 2 and 5 years ago (medium); and iv) less than 2 years ago (new). Also, for each question, participants should provide their academic and industrial experience. This scheme is important as the experience of participants may impact the results of the experiment. According to Falessi et al. [7], conducting experiments with students does not have a lower relevance or lower interest than experiments with professionals. In the same context, Feldt et al. [8] argue that researchers should discuss the limitations of having students as participants while aiming to run more experiments with practitioners. In our experiment, 19 out of the 24 participants have varied professional experiences, being both students and practitioners. Details about the experiment design, including the sample selection, are described in the next section.

3 EXPERIMENT DESIGN

In this section, we present the experiment design[1] defined based on the aforementioned related work.

3.1 Experiment Definition and Planning

The goal of our experiment is *to analyze if the provision of RRs brings benefits to developers*. Moreover, we want to observe how developers perceive the difference between using RRs compared to not using them. We also aim at investigating if developers have preferred RRs when fixing inconsistent models. Thus, the goal of the experiment is not to compare two RRs techniques to conclude which one is better. Rather, we aim at understanding and evidencing the real benefits (if they exist) of providing RRs from the perspective of the developers. This clarification is important, as the experiment was planned and executed with this goal in mind.

3.1.1 *Research Questions, Metrics, and Hypotheses.*

The independent variable is the use of RRs for repairing design models. We decided to use UML diagrams to represent the models since UML is a well-known modeling language adapted in the industry for documentation [3], as well as being commonly used in controlled experiments [5, 6]. The values (treatments) given to this variable are: repair recommendations (RR), providing a set of repair alternatives for each inconsistency; and without recommendations (WR), repairing the diagrams only knowing their inconsistencies. The dependent variables are: Time spent (in minutes) on the tasks ($\mathbb{T}$); the number of inconsistencies per task ($\mathbb{I}$); the number of inconsistencies fixed per task ($\mathbb{IF}$). We attribute different $\mathbb{IF}$ scores as an inconsistency could be fixed (score 1), partially fixed (score 0.5), or not fixed (score 0); Additional metrics include the ranking of repair recommendations given by the participants in RR tasks ($\mathbb{RRR}$); repairs created by participants in WR tasks (RC). In addition, we use four qualitative metrics, presented in Table 1: task difficulty level ($\mathbb{TD}$) and overall difficulty ($\mathbb{OD}$); confidence level ($\mathbb{CL}$) regarding the inconsistency being repaired; and the participants' acceptance ($\mathbb{PA}$) regarding RRs.

We also formulate indirect metrics based on related work [21], namely, effectiveness (EFT) and efficiency (EFY). EFT is the degree to which participants repair a model inconsistency (Equation 1). The result is a range from 0 to 1, where 0 means that no participant repaired the inconsistency and 1 means that all participants repaired the inconsistency for that task. EFY is the degree to which participants repair a model inconsistency, considering the time required (Equation 2).

$$EFT = \frac{\mathbb{IF}}{\mathbb{I}} \quad (1) \qquad EFY = 60 * \frac{EFT}{\mathbb{T}} \quad (2)$$

We also complement the analysis of the results by extracting themes from open-ended questions present in the feedback questionnaires.

Table 1: Questions from the experiment questionnaires

ID	Question	When it was asked	Type
𝕋𝔻	How difficult was this task?	After each task	5-Point Scale
ℂ𝕃	How confident are you about fixing the inconsistency in this task?	After each task	5-Point Scale
𝕆𝔻	How difficult was repairing the models (RR and WR)?	After all tasks	5-Point Scale
ℙ𝔸	Do you think that using RRs is worse or better than not using them?	After all tasks	5-Point Scale
ℂ𝔽	What challenges did you face when repairing the models (RR and WR)?	After all tasks	Open-ended

These questions included asking participants about the challenges faced (see ℂ𝔽 in Table 1) when performing the tasks, justifying the reason why an inconsistency was not fixed, and how the given RRs could be improved in terms of understandability. From all the answers given, we applied a thematic synthesis process to create and extract themes [4], describing patterns that are found in the answers. These metrics are used to answer the following RQs:

RQ1. Do developers benefit from recommendations when repairing inconsistent design models?

Rationale: we investigate if the provision of RRs impacts the effectiveness and efficiency of the participants when repairing inconsistent models. **Method:** we answer this RQ by testing two-sided hypotheses for the EFT and EFY metrics. The null hypothesis H_0 states that the provision of RRs has no impact, as it does not increase or decrease the EFT/EFY compared to WR. The alternative hypothesis H_a states that RRs have an impact on the EFT/EFY scores. To test the statistical significance of the results, we performed a two-tailed distribution, paired t-test, with a probability level of 5% to reject H_0:

$$H_{0EFT} : EFT(RR) = EFT(WR) \quad H_{0EFY} : EFY(RR) = EFY(WR)$$
$$H_{aEFT} : EFT(RR) \neq EFT(WR) \quad H_{aEFY} : EFY(RR) \neq EFY(WR)$$

RQ2. How do developers perceive the use of repair recommendations when repairing inconsistent design models?

Rationale: we investigate participants' opinions regarding the difficulty of the tasks performed, their confidence when performing the tasks, and the acceptance of using RRs when repairing inconsistent models. We analyze these results to understand if when RRs are provided, participants perceive tasks to be less or more difficult while having less or more confidence in performing them. **Method:** we consider four questions from the feedback questionnaires given to participants, namely, 𝕋𝔻, ℂ𝕃, 𝕆𝔻, and ℙ𝔸, shown in Table 1. We also consider the two metrics used for the hypothesis testing performed for RQ1 when analyzing the correlation between the perceived difficulty and the effectiveness and efficiency results.

RQ3. Do developers have preferred recommendations when repairing inconsistent design models?

Rationale: we investigate if there is one ideal RR for each inconsistency. We also investigate the possibility of RRs that are always or never selected. **Method:** we consider the RRs ranking assigned by participants in each RR task. Then, we analyze the frequency of an RR being selected as the first option as well as the most frequent, best, and worst rank of each RR. For WR tasks, we look at the repairs created by participants, comparing them with the RRs provided for the RR tasks.

Table 2: Experience of the 24 Participants

Experience Level	Number of Participants (Academic/Industrial Exp.)			
	Soft. Eng.	UML	Soft. Repair	UML Repair
None	1/5	1/15	6/9	11/21
Short	4/8	10/7	7/6	12/3
Medium	14/8	12/2	9/8	1/0
Long	5/3	1/0	2/1	0/0

3.1.2 *Sample Selection.*

We invited participants for the experiment using convenience sampling, i.e., participants selected were easily accessible [27]. We sent e-mail invitations to M.Sc./Ph.D. students enrolled in the Model-Driven Engineering course. After the students replied, demonstrating interest in participating in the experiment, we sent a questionnaire to collect demographic data about them. This questionnaire to characterize participants[1] was designed using the R^3 scheme (Real, Relevant, and Recent) considering both academic and industrial experience [7]. The strategy for inviting participants and collecting their data was designed based on the mitigation strategy of threats to validity (TtV) TtV1 and TtV2, described in Section 3.3.

In total, 31 participants answered the experience questionnaire, however, seven participants were not able to be present in the experiment. Hence, the sample consists of 24 participants (22 M.Sc. and two Ph.D. students). As shown in Table 2, the experience of the participants varied between the categories of short, medium, and long, with most participants having at least short experience in the industry (19) with software engineering. We only invited participants that were familiar with UML, since all the tasks given contained UML diagrams. Table 2 also shows that only one participant did not have experience with UML in the academy, however, the same participant had a short experience with UML in the industry.

3.1.3 *Experimental Package.*

The experimental package was designed considering the mitigation of threats to validity (see Section 3.3) and it was composed of the following artifacts:

Design models: UML diagrams that are part of a dataset of both industry and academic models of different domains.[1] This dataset has also been used for evaluating RR generation approaches [12, 16, 23]. We used six different UML models, composed of class, sequence, and state machine diagrams.

Guidelines: this document included an explanation about consistency checking and RRs, how the tasks should be conducted, and the consent term that participants had to agree on. The guidelines also contained two illustrative tasks that were completed by participants together with an experimenter carrying out the session. The definition of the guidelines was based on the mitigation from TtV5.

Tasks' document: the tasks were given to participants as a printed document. An example of a task is illustrated in Figure 1. For each task, we presented the CR applied and described the inconsistency with a sentence in English (at the top of the figure). We also highlighted in the diagram the inconsistency to be fixed (the three ellipses). To ensure that the inconsistencies given in the tasks were varied, for some tasks we introduced the inconsistency,

Table 3: Experiment Tasks

Group	Task	Model	Diagram	CR	Inconsistency
G1	TCex	M1	class and seq.	CR1	Original
	TCac	M2	class and seq.	CR2	Introduced
G2	TMic	M3	class	CR3	Introduced
	TCdr	M4	class	CR4	Original
G3	TSeq	M4	seq. and state	CR5	Introduced
	TSta	M4	class and state.	CR6	Introduced
G4	TCno	M5	class and seq.	CR1	Original
	TSup	M6	class and seq.	CR1 & CR2	Introduced

i.e., by deleting or modifying an element from the diagram. We defined five task criteria to be followed when creating a task: the diagram used in the task must be of a certain size range, between 3 and 15 model elements; tasks should have different combinations of diagrams, e.g., only class, class and sequence, class and state machine, and sequence and state machine; each task must have unique inconsistencies; an inconsistency must have multiple RRs; inconsistencies must affect different types of model elements, e.g., operations, associations, messages, lifelines, and states.

Table 3 summarizes the description of the eight tasks defined, six main tasks (G1 - G3), and two extra tasks (G4). These two extra tasks were defined based on feedback from the pilot studies conducted before the experiment (details in Section 3.3, TtV7). Tasks were grouped in pairs based on the diagram types being used, e.g., G3 is composed of two tasks that contain state-machine diagrams. The reason for grouping tasks is due to the order of the tasks changing for each participant (see Section 3.2).

Repair recommendations: The RR used in the experiment were generated by adopting approaches and applying them into the inconsistent models [12, 16, 22]. During the experiment, each participant received RRs for four tasks (one task per task group), having to perform the other four without RRs. For instance, Participant 1 received RRs for TCex but not for TCac, which belong to G1. Participant 2, however, received RRs for TCac and not for TCex. We ensured that no matter the order of the tasks given to each participant, they would alternate between RR and WR. When a task contained RRs, participants were asked to write down the rank for selecting each RR. For instance, as illustrated in Figure 1, five RRs were given, hence, participants rank them using numbers (from 1 to 5) to determine which RR is more suited in their opinion (1 being the most suited one). They could also use the same rank more than once if they thought that two or more RRs were equivalent, e.g., *2. Modify* and *4. Modify* could be both considered the first option. The participants could also write down that one or more RRs were non-applicable for that task by writing "N/A" or 0 (zero). Lastly, participants could include new RRs if they believed that none of the RRs was valid.

Feedback questionnaires: after the completion of each task, participants were asked to answer a feedback questionnaire composed of questions $\mathbb{TD}$ and $\mathbb{CL}$ from Table 1. After the completion of all tasks, participants were given an overall feedback questionnaire with questions $\mathbb{OD}$, $\mathbb{PA}$, and $\mathbb{CF}$ from Table 1 among others.[1]

3.2 Operation of the Experiment

The experiment was carried out in a classroom at the university on two different dates due to the availability of participants. On the first date, fourteen participants were present, and ten participants were present on the second date. For both dates, the experiment operation was carried out following the same script, with the same three experimenters in the room. The experiment started at 10:35 AM on both dates and lasted (the time when the last participant handed in the feedback questionnaire) until 11:35 AM on the first date and 11:18 AM on the second date. At the start of the experiment, participants were given an explanation about the experiment's main goal, without mentioning RQs or hypotheses. The guidelines document was given to each participant and one experimenter read it aloud also asking participants if they had any questions related to it. At this point, we explained that the tasks must be performed individually and that participants were allowed to ask questions. However, questions related to how to create or rank repairs would not be answered to avoid bias. Furthermore, participants were not allowed to talk with other participants. We also reminded participants to write down the current time before and after each task.

After all the participants had finished reading the guidelines, the task document was given to them. While participants were performing the given tasks, three experimenters were in the room observing if participants were following the guidelines explained. In total, we had 24 different task documents printed, one for each participant. All documents had the same eight tasks (see Table 3) and the same feedback questionnaire for each task. However, the order and the type of the tasks changed for each document, such as that participants would alternate between RR and WR tasks. This structure is important to mitigate a threat to validity related to the learning effect (TtV6 in Section 3.3). Once a participant have finished their tasks, the experiment feedback questionnaire was given. After the experiment, the documents filed by the participants were converted into raw data.

3.3 Threats to Validity

In this section, we discuss threats to validity (TtV) and how we mitigated them.

Construct Validity: The first threat is using a convenience sample, as it can bias the results based on the participants' experience (TtV1). To mitigate this, we applied a questionnaire with the R^3 characterization mechanism [7] to collect data about the academic and industrial experience of participants on different topics. This experience was considered when analyzing the results of each participant. Furthermore, the majority of participants (19 out of 24) have industrial experience. However, collecting the background data about the participants may have a negative impact, such as the stereotype threat (TtV2), which may lead to less experienced participants having less confidence as they can be reminded of their lack of experience [11, 17]. We mitigated this threat by collecting participants' demographic data through an online form, one week before the experiment. The options provided as possible answers to close-ended questions are also a threat, as they may confuse or unintentionally bias participants' answers (TtV3) [17]. Hence, when using the Likert scale for answers, we always used textual

descriptions for all options given. We also included open-ended questions, asking participants to include observations.

Internal Validity: The models selected may bias the results of the RQs (TtV4). The models used in our experiment, however, were already evaluated and used in other studies [12, 23], being thoroughly tested. Furthermore, the dataset was composed of both industry and academic models with three different types of UML diagrams and combinations of them for the tasks, as described in Table 3. Another threat is related to participants not reading or understanding the instructions (TtV5). To mitigate this, participants were asked to silently read the guidelines while an experimenter read the same guidelines aloud. The guidelines also contained two illustrative tasks that simulated the two types of tasks (RR and WR).

Conclusion Validity: A threat is related to the tasks' order and structure, impacting the results (TtV6) due to the learning effect [11, 24]. To mitigate this threat, we structured the experiment tasks using a counterbalancing strategy [11, 26]. Hence, the participants alternated between RR and WR tasks with the order changing, such as that no two participants did the same combination of tasks (RR or WR) in the same order.[3] Another threat is related to task difficulty and completion time (TtV7) that impact the results by creating fatigue in the participants [11]. We mitigated this by conducting two pilot studies with four participants (two for each pilot) that were not present in the final sample. After each pilot study, we interviewed these participant. Based on their feedback for each task regarding difficulty and the time spent, we re-balanced the tasks and time required for the experiment operation. For instance, the first pilot study had only six tasks. Participants from the pilot studies also reported that most tasks were easy, so we re-balanced these tasks, increasing their difficulty. Measuring the success of WR tasks can also be a threat to validity, since incorrect measures may impact the WR tasks' results (TtV8). To mitigate this, we checked if the RRs created by participants in WR tasks were part of the generated RRs from the same RR task. Lastly, measuring the time wrongly due to participants not being sure when the task started and finished may threaten the results regarding efficiency (TtV9). We asked participants to only start the tasks after signing the consent form agreement to mitigate this threat. Whenever starting a task, they would write down the start time at the top of the page. They also recorded the end time before answering the feedback questionnaire for each task. To keep the time consistent, we asked participants to use the time from a digital clock displayed on the beamer.

External Validity: The impossibility to generalize the results to practical cases due to the participants of the experiment being students is a threat (TtV10). To mitigate this threat, we only invited participants that already concluded their bachelor's degree. As shown in Table 2, the majority of the participants (19 out of 24) had experience in the industry. Furthermore, we argue that even inexperienced developers should be able to use RRs. The last threat is related to the experimental package not representing a practical case, such as using toy examples that are not representative of industrial practice (TtV11). To mitigate this, we used models that were taken from real systems and the diagrams were scaled down to be of a manageable size. However, as all elements of the model that were relevant to each inconsistency were kept in the diagram, we argue that scaling down the models would not affect the inconsistency repair.

[3]The definition of the task order is presented in our online appendix [15].

4 RESULTS AND DISCUSSION

In this section, we present the results of the experiment, and also discuss the limitations and lessons learned.

4.1 Answering the RQs

RQ1. Do developers benefit from recommendations when repairing inconsistent design models?

Table 4 describes the results of the effectiveness and efficiency metrics per task, distinguished by task type (RR and WR). Considering the effectiveness result, column *EFT*, the use of RR always led to 1 effectiveness (100%). This means that for RR tasks, participants always selected a repair to fix the given inconsistency. For the WR tasks, however, the result was 1 (100% EFT) only for task TSta. The lowest result was for task TCex WR, where participants obtained 0.29 effectiveness on average. This means that inconsistencies were fixed (three in total) or partially fixed (one in total) in TCex WR for only 29% of the participants.

The effectiveness results of task TCex cannot be attributed to the order in which TCex was performed, since the order of the tasks changed for each participant. Similarly, tasks TCno and TSup were always the last two tasks, still, the effectiveness scores for these tasks were among the worst. We observed that the reason for these lower scores is the inconsistency type and their required RRs, which were suggesting modifying the association between UML classes. Such RRs are more complex than *renaming* or *deleting* model elements that were part of WR tasks that obtained the best effectiveness results, i.e., tasks TCac, TMic, TCdr, TSeq, and TSta. Thus, the results regarding effectiveness indicate that *participants can benefit from RRs in more complex tasks by improving the effectiveness while keeping similar effectiveness for simpler tasks.*

Figure 2 illustrates the results related to the time ($\mathbb{T}$) required (in minutes) to complete each task. The average time is similar for RR and WR tasks. We observed that in the RR tasks, participants spent their time analyzing the RRs given and their respective impacts on the model. Whereas in WRs tasks, participants used the time to think about which changes would possibly fix the given inconsistency and how to formulate them. In cases where simple changes, such as *renaming*, were sufficient to fix the inconsistency, participants spent less time on WR tasks. This is evidenced by the completion time results of tasks TCac, TSeq, and TSta.

In the tasks where more complex changes were required, such as TCex and TCno, time greatly varied when RRs were not given. This is shown by analyzing the interquartile range regarding the completion time for these two WR tasks, ranging from around 4 to 8 minutes for task TCex and from 2 to 7 minutes for task TCno (see Figure 2). This range is much larger in comparison with the same range for task TCex RR (from 3 to 5 minutes) and TCno RR (from 2 to 4 minutes). The difference in these ranges indicates that *the time required to fix inconsistencies varies more when repairing more complex inconsistencies without RRs.* Due to the different completion times on average, which is shown in column $\mathbb{T}$ in Table 4, five tasks (TCex, TMic, TCdr, TCno, and TSup) were performed with more efficiency when RRs were given. In the other three tasks (TCac,

Table 4: Results of EFT and EFY per task and overall

Task	Type	$\mathbb{I}$	$\mathbb{IF}$ (avg.)	EFT	Avg. $\mathbb{T}$ (min)	EFY
TCex	RR	1	1.00	1.00	3.92	15.31
	WR	1	0.29	0.29	5.58	3.14
TCac	RR	1	1.00	1.00	3.33	18.02
	WR	1	0.92	0.92	2.92	18.84
TMic	RR	1	1.00	1.00	2.83	21.20
	WR	1	0.96	0.96	3.00	19.17
TCdr	RR	1	1.00	1.00	3.17	18.93
	WR	1	0.92	0.92	3.58	15.36
TSeq	RR	1	1.00	1.00	4.50	13.33
	WR	1	0.92	0.92	4.08	13.48
TSta	RR	1	1.00	1.00	3.92	15.31
	WR	1	1.00	1.00	2.58	23.26
TCno	RR	1	1.00	1.00	3.42	17.54
	WR	1	0.38	0.38	3.92	5.74
TSup	RR	2	2.00	1.00	3.42	17.54
	WR	2	0.88	0.44	3.92	6.70

Metric	Type	Mean	Median	Std.	% dif.	p-value
EFT	RR	1.00	1.00	0.00	37.63	0.04
	WR	0.73	0.92	0.30		
EFY	RR	17.15	17.54	2.45	29.81	0.17
	WR	13.21	14.42	7.29		

TSeq, and TSta) participants obtained higher efficiency results when no RRs were given. Thus, *RRs benefit participants in complex tasks by improving efficiency*. For *simpler tasks*, however, *the provision of RRs reduces the efficiency of participants*.

Table 4 shows the results of the hypotheses testing, comparing the mean, median, and standard deviation of the effectiveness (EFT) and efficiency (EFY) metrics. Column *% diff* shows the percentage difference between the two means, i.e, a positive value means that the use of RRs is better in terms of effectiveness or efficiency, whereas a negative value means that RR is worse (column *p-value*). Notice that the result for effectiveness is statically significant (p-value < 0.05) while for efficiency it is not statically significant.

Answering RQ1: *The use of RR has an impact as it improves EFT by 37.63% on average, rejecting H_{0EFT} (p-value = 0.04). The use of RRs also improves EFY by 29.81% on average (p-value = 0.17). Providing RRs, however, decreases the EFY of developers in simpler tasks compared to manual repairs.*

RQ2. How do developers perceive the use of repair recommendations when repairing inconsistent design models?

Considering the results of tasks difficulty ($\mathbb{TD}$) for four tasks (TCac, TSeq, TSta, and TCno), the majority of participants had similar answers for WR and RR tasks with most being either *easy* or *neutral* difficulty. For the other tasks, answers were more distributed. For instance, for task TCex RR, 58% of the participants perceived the task as having *neutral* difficulty, while for TCex WR the task was *difficult* for 42% and *neutral* for 42% of the participants. For TMic RR, 83% of the participants considered the task to be *easy*. Considering TMic WR, however, the answers were varied, with only 25% of the participants considering the task to be *difficult*. Considering the question about difficulty asked after all tasks ($\mathbb{OD}$), the majority of participants (50%) considered the RR tasks to be *easy*. The answers for WR tasks, however, were the same for *neutral* and *difficult*, with 42% of the participants choosing either option. Hence, we observe that the *participants' perceived difficulty per task is not impacted by the provision of RRs*, instead, the participants' *perceived difficulty is related to the complexity of the inconsistency and the possible RRs required.*

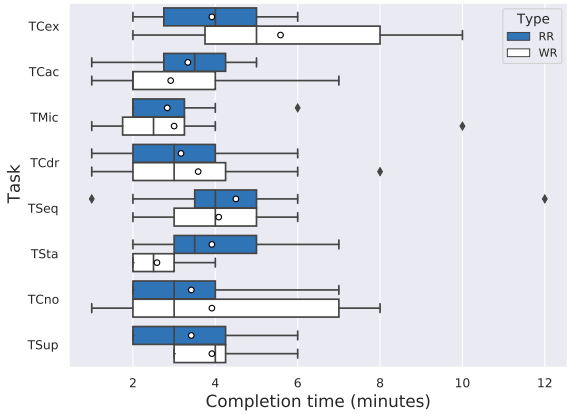


Figure 2: Result of time (in minutes) spent on tasks

The confidence level ($\mathbb{CL}$) of participants about fixing the inconsistencies cannot be related to the provision or the lack of RRs since, for most tasks, the results were similar.[4] An exception is TSeq, where the majority of participants (42%) were *fairly confident* without RRs, representing the number four in the 1-5 scale. Moreover, for RR tasks, 50% of the participants were *somewhat confident*, representing the number three on the 1-5 scale.

The results for the confidence level can be related to the difficulty level results, since tasks considered to be easier were those where participants felt more confident (TCac, TMic, TCdr). Furthermore, tasks considered to be more difficult were those where the $\mathbb{CL}$ was lower (TCex, TCno, and TSup). Hence, *the provision of RRs does not impact the confidence level of participants when fixing inconsistent models*. These results can be complemented by analyzing two themes extracted from the open-ended questions. In these questions, twelve participants reported that, for some tasks, *it was hard to repair the inconsistency due to their lack of knowledge about the model's domain*. This is also related to *how inconsistencies are presented and described to the developers*, which was reported by two participants as impacting their decision. Providing this information may be useful when deciding on RRs, especially considering RRs that should not be applied since they may contradict the original design of the model.

Concerning the participants' acceptance of RRs ($\mathbb{PA}$), the majority of participants (88%) answered that using RRs when fixing inconsistent models is either *much better* or *somewhat better*. The option *about the same* was chosen by 4%, and *somewhat worse* selected by 8% of the participants. The option *much worse* was not selected by any participant. In addition to these results, if we consider the themes extracted from open-ended questions, ten participants mentioned that *the lack of RR made some tasks harder*. These results indicated that *participants feel that the use of RRs is better than not using them* when repairing models.

[4] Results of all questions are available at our online appendix [15]

Table 5: Results of RRs ranking

Task	RR	Ranked 1st*		Overall Ranking					
		#	%	Avg.	Std.	Mode	Best	Med.	Worst
TCex	RR1	9	75	1.25	0.45	1	1	1	2
	RR2	0	0	3.25	0.62	3	2	3	4
	RR3	4	33.33	2.08	1.08	2	1	2	4
TCac	RR1	3	25	2.25	1.06	2	1	2	4
	RR2	9	75	1.58	1.16	1	1	1	4
	RR3	1	8.33	2.92	1	3	1	3	4
TMic	RR1	1	8.33	4.67	1.56	6	1	5	6
	RR2	0	0	3.83	1.59	2	2	3.5	6
	RR3	9	75	1.42	0.79	1	1	1	3
	RR4	1	8.33	2.58	1.31	2	1	2	6
	RR5	0	0	4.92	1.51	6	2	5.5	6
TCdr	RR1	2	16.67	3.83	1.64	5	1	5	5
	RR2	10	83.33	1.25	0.62	1	1	1	3
	RR3	0	0	4.25	1.22	5	2	5	5
	RR4	8	66.67	1.50	0.90	1	1	1	4
TSeq	RR1	4	33.33	2.58	1.51	3	1	3	6
	RR2	3	25	3	2.09	2	1	2	6
	RR3	3	25	2.67	1.23	4	1	3	4
	RR4	2	16.67	3.42	1.78	3	1	3	6
	RR5	2	16.67	4.42	1.98	6	1	5	6
TSta	RR1	2	16.67	3.08	1.88	2	1	2.5	6
	RR2	6	50	1.92	1.44	1	1	1.5	6
	RR3	3	25	3.58	2.11	6	1	3	6
	RR4	3	25	3.67	2.06	6	1	4	6
	RR5	0	0	5.42	1	6	3	6	6
TCno	RR1	2	16.67	2.67	1.07	3	1	3	4
	RR2	3	25	2.25	1.06	2	1	2	4
	RR3	7	58.33	2.08	1.44	1	1	1	4
TSup	RR1	5	41.67	2.75	1.86	1	1	2	5
	RR2	3	25	2.33	1.15	2	1	2	5
	RR3	5	41.67	2.42	1.56	1	1	2	5
	RR4	3	25	3.50	1.73	5	1	4	5

*-Percentage is based on the number of participants that ranked RRs 1st. Since more than one RR could be ranked 1st, percentages may sum up to more than 100%

Answering RQ2: *The provision or the lack of RRs has no direct impact on participants' perceived difficulty and confidence. Also, participants feel that using RRs is better than not using them.*

RQ3. Do developers have preferred recommendations when repairing inconsistent design models?

Table 5 shows the results for the RR ranking (ℝℝℝ) for each RR task. Column *Ranked 1st #* shows the number of times that an RR was selected as the first option. For each RR task, 12 participants had to select RRs, since the other 12 participants performed the same task without RRs. For task TCex, RR1 was the first option for 9 participants (75%) while RR2 was never the first option, and RR3 was the first option for 4 participants (33.33%). Because participants could repeat the ranking for two or more RRs, the sum of the percentages can be greater than 100%. None of the tasks had only one RR always selected as the first option. In four tasks (TCex, TCac, TMic, and TCno), only one RR was the first option for more than 50% of the participants. In three tasks (TSeq, TSta, and TSup), no RR was selected by more than 50% of the participants. For task TCdr, two RRs were selected by more than 50% of the participants. There were some cases, such as RR1 and RR3 for TSup, where two RRs were selected as the first option by the same amount of participants. The results indicate that *there is not one ideal solution for repairing an inconsistency, but rather participants prefer different RRs.*

Table 5 also shows results for the overall ranking of RRs per task. The average ranking (column *Avg.*) shows that, for most tasks, at least one RR stayed close to the bottom rank. This means that such an RR is always considered not applicable for the given inconsistency. Examples include RR3 for task TCex (avg. ranking of 3.25) and RR5 for TSta (avg. ranking of 5.42). For both RRs, the ranking given was greater than the number of options, i.e., for task TCex only three RRs were given and RR3 ranked 3.25 on average. This happens because when an RR was not selected, i.e., not ranked, we considered it to be the worst possible rank. For task TCex, the worst possible ranking was 4th, since three RRs were given. Thus, the 4th ranking for task TCex means that this RR was not selected. Tasks TSta, TCno, and TSup had RRs that were, at least once, not selected. This is displayed by the column *Worst* in Table 5, which describes the worst ranking given for each RR. These results indicate that, *depending on the context, any RR may be considered not applicable.* The context in this case is the inconsistency, the broken model, and the personal preferences of the developer. Further support for this finding is given by analyzing three themes from the open-ended questions. Firstly, nine participants mentioned that *some RRs seemed not suited to fix the inconsistency.* Most of these RRs mentioned were the ones that suggested deleting elements. Secondly, eleven participants (45%) commented that *the lack of side effects analysis related to the application of RRs made the decision more difficult.* Hence, if such analysis is provided, the decision of using or not an RR may be impacted. Lastly, one participant mentioned that a *better rationale for using the RRs should be provided.* These three themes indicate that only providing a list of RRs may not be sufficient for developers. Approaches should instead provide enough rationale related to why each RR should be considered and what are their side effects.

Considering the best ranking in Table 5 (column *Best*), for most tasks all RRs were considered as the first or second option, at least for one participant. The only exception, task TSta, had one RR (RR5) which had 3rd as the best ranking. The most frequent ranking (column *Mode*) for this same RR was 6, which means that this RR was most frequently not selected at all. If we consider the results for the other seven tasks, excluding TSta, we observe that *any RR may be considered as the most suited alternative.*

Table 6: Repairs Created (RC) in WR tasks

Task	RC	#	%	Rel. RR	Task	RC	#	%	Rel. RR
TCex	RC1	3	25	RR1	**TCno**	RC1	1	8.33	-
TCac	RC1	1	8.33	RR3	**TCdr**	RC1	6	50	RR2
	RC2	10	83.33	RR2		RC2	8	66.67	RR4
						RC3	3	25	RR1
						RC4	2	16.67	RR3
TMic	RC1	11	91.67	RR3	**TSeq**	RC1	3	25	RR1
	RC2	1	8.33	-		RC2	6	50	RR2
	RC3	1	8.33	RR2		RC3	3	25	RR4
TSta	RC1	2	16.67	-	**TSup**	RC1	7	58.33	RR3
	RC2	8	66.67	RR2		RC2	8	66.67	RR1
	RC3	5	41.67	RR4		RC3	1	8.33	RR2

Table 6 describes the repairs created (RC) by the participants in the WR tasks. We analyzed all WR tasks and counted the number of times (column #) that the same RC was performed. While performing this analysis, we only considered RC that fixed the inconsistencies in the task. Tasks TCex and TCno had only one RC each, these are the same tasks with the lowest effectiveness and

efficiency results. The repair for task TCex was used only for three participants, representing 25% of the participants that performed TCex WR. The RC for task TCno was only performed by one participant. We also noticed that most repairs created for task TCno (seven cases) only partially fixed the inconsistency in the task. For both TCex and TCno, the number of participants able to formulate a correct repair stayed below four. This indicates that *the lack of RRs can lead to few repairs being considered as alternatives.*

The columns # and % from Table 6, show that in four tasks (TCac, TMic, TCdr, and TSta) only one RC was used for more than 50% of the participants. In one task (TSup) two RCs were used for more than 50%. One important aspect observed is that for all tasks, except TCdr, participants did not consider all possible repairs (the ones given in RR tasks). Additionally, the columns # and % from Tables 5 and 6 can be compared to observe that *when RRs are provided, participants considered using different repair alternatives.* This is further supported by a theme extracted from the comments of two participants, mentioning that *some RRs alternatives were not considered when RRs were not given.*

The column *Related RR* in Table 6 shows RRs that were part of the RR tasks, which proposed the same changes represented by the RCs. Only in three cases, the repairs created by participants did not have a corresponding RR. For all three cases, this happened because the RCs suggested deleting the model element which contained the inconsistency. Two of these three RCs were applied only once (RC2 for TMic and RC1 for TCno) and the other was applied by two different participants (RC1 for TSta). Although these RRs were generated by the approach, based on the feedback from the pilot experiments (see Section 3.3), we decided to remove such RRs from the experiment. The reason was that participants from the pilot described deleting the model elements that *cause* the inconsistency to be *"senseless"* RRs. Thus, *the repairs created by participants were also proposed by the approach.*

Answering RQ3: *There is not one ideal RR for repairing a given inconsistency. Depending on the context, the developer's preferences change as any given RR may be considered the ideal one. There are, however, contexts where any RR may be considered not applicable. Furthermore, when RRs are provided, developers consider more alternatives to fix the inconsistency.*

4.2 Lessons Learned and Limitations

In this section, we discuss the lessons learned with the results and the limitations of this study.

RRs in simpler tasks: Considering RQ1, the results of efficiency were not statistically significant since for three tasks (TCac, TSeq, TSta) the efficiency was worse when RRs were given. These results led us to understand that providing RRs may not be always the best solution for all tasks. Tasks TCac, TSeq, and TSta had inconsistencies that could be fixed with simple changes such as *renaming* a model element. In these cases, participants can create their repairs fairly fast. When RRs are given, however, participants need to analyze all alternatives before selecting, increasing the required time. This analysis is important and not addressed in other studies, as it is always assumed that *using repairs is better than not using them.* The results of our experiment show that this is not always the case, as *in simpler tasks the use of RRs did not affect the effectiveness* and *providing RRs reduced the efficiency in simpler tasks.* This is an important lesson learned, as we observed that it may not be required to provide repairs for all inconsistencies when repairing models, as simpler inconsistencies can be easily repaired manually. Deciding which inconsistencies are "simpler", however, is not trivial and is related to the developers' knowledge and experience. Since our experiment is limited to the types of models and inconsistencies provided, further investigation has to be carried out on this topic. The results of this investigation may aid practitioners to develop approaches that are flexible to the developers/model context.

Relation of effectiveness/efficiency with difficulty and confidence: The results of the difficulty and confidence of participants (RQ2) can be related to the effectiveness and efficiency scores from Table 4 (RQ1). This relation is important since low effectiveness and efficiency may be associated with the difficulty of the task. The three tasks considered more difficult by participants (TCex, TCno, and TSup) are the same below 0.5 effectiveness and below 10 efficiency. Furthermore, tasks TCac, TMic, and TCdr were considered either *easy* or *very easy* and stayed between 0.9 and 1 effectiveness and between 10 and 20 efficiency. Similarly, the confidence level may also be associated with the effectiveness and efficiency results. In two RR tasks (TCex, TCno), although participants were always able to fix the inconsistency (effectiveness = 1), they still did not feel confident about it. These results indicate that, *in some cases, RRs still seem wrong in the developers' perception as they are not confident that the inconsistency is fixed by these RRs.* According to the answers to open-ended questions, some participants reported that they needed additional information from the RR when deciding the rank. This information could include the impact of the RR in the model, i.e., what the model would look like after the RR has been performed. Additional information, according to the participants, would be to include previous changes from the model history. This information would help the developers understand how the model was designed and what was the intent of the designer. Our results are limited in this regard, as we did not include this information and focused on the current state of the models only. Analyzing these implications remains as future work.

RRs preferences: The results related to developers' confidence are also important to better understand the impact of RRs from the developers' perspective, including *how much* the developer *trusts* the given suggestion. This analysis relates to the use of automatic repair approaches which, usually, do not rely on developers' feedback [13]. These approaches produce correct repairs, however, those repairs may not be *correct* or the *ideal* alternative in the eyes of the developer. This argument is supported by the results of RQ3, which show that developers have different preferences when selecting repairs. Similarly, designing an approach that ranks the RRs alternatives brings challenges, as even the rank is different depending on developers' preferences. Moreover, some RRs are preferable depending on different types of models and inconsistencies. Hence, the ranking of RRs should consider the context in which it should be applied. We argue that *when ranking RRs, approaches should take into account the preferences of developers and should be context-sensitive.* Our results, however, cannot support the claim regarding the context as all models and inconsistencies used derive from the same domain (UML) and further investigation is needed.

RRs generated: The analysis of the results is limited by the RRs generated because a different RR approach for generating or presenting the RRs may lead to developers ranking or selecting different RRs. This could also impact other metrics such as efficiency, $\mathbb{TD}$, and $\mathbb{CL}$. Hence, *if a different approach was used for generating the RRs, the results may differ*. Due to this limitation, we decided to give the tasks to developers using pen and paper, instead of relying on a tool. By using this strategy, we allow the protocol of the experiment to be easily adapted by replacing the approach used for generating the RRs as the protocol is tool-independent. This strategy facilitates the replicability of the experiment to confirm or decline the findings using different approaches. We understand, however, that the findings and answers for the RQs can only be supported considering the approach applied in this experiment.

5 CONCLUSION

The benefits of using RRs when repairing inconsistent design models have never been investigated in an experiment with human participants. In this work, we reported a controlled experiment carried out with 24 developers. The results evidence that developers benefit from RRs by improving the effectiveness and efficiency when repairing inconsistent design models. This improvement is more evident when the tasks are complex. We also observed, however, that RRs can reduce efficiency in simpler tasks. Moreover, the perceived difficulty and confidence of developers are not impacted by the provision of RR. Furthermore, we concluded that not all developers choose the same RR, but rather, have varied preferences. Hence, the provision of RRs leads to developers considering more alternatives for fixing an inconsistency.

We also reported lessons learned, limitations, and future work planned to address them. For instance, improving the RRs feedback given to developers by exploring the side effects of repairs. We argue that this aids developers during the decision-making process. However, these potential implications should be further explored in studies focusing on side effects and RRs rationale. The investigation of the benefits and challenges of side effects and change propagation of RRs from the developers' perspective can aid researchers and practitioners to formalize the repair process for design models.

6 DATA AVAILABILITY

Artifacts of the experiment are available in an online appendix [15].

ACKNOWLEDGMENTS

This research has been funded by the Austrian Science Fund (FWF, P31989-N31), by the FFG-COMET-K1 Center "Pro^{2}Future", (881844), and by the LIT Secure and Correct System Lab funded by the State of Upper Austria.

REFERENCES

[1] Angela Barriga, Rogardt Heldal, Adrian Rutle, and Ludovico Iovino. 2022. PARMOREL: a framework for customizable model repair. *SoSym* (2022), 1–24.
[2] Raja Sehrab Bashir, Sai Peck Lee, Saif Ur Rehman Khan, Victor Chang, and Shahid Farid. 2016. UML models consistency management: Guidelines for software quality manager. *IJIM* 36, 6, Part A (2016), 883–899.
[3] Federico Ciccozzi, Ivano Malavolta, and Bran Selic. 2019. Execution of UML models: a systematic review of research and practice. *SoSym* 18, 3 (2019), 2313–2360.
[4] Daniela S. Cruzes and Tore Dyba. 2011. Recommended Steps for Thematic Synthesis in Software Engineering. In *ESEM*. 275–284.
[5] África Domingo, Jorge Echeverría, Óscar Pastor, and Carlos Cetina. 2021. Comparing UML-based and DSL-based Modeling from Subjective and Objective Perspectives. In *CAiSE*. Springer, 483–498.
[6] Wojciech J. Dzidek, Erik Arisholm, and Lionel C. Briand. 2008. A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance. *IEEE TSE* 34, 3 (2008), 407–432.
[7] Davide Falessi, Natalia Juristo, Claes Wohlin, Burak Turhan, Jürgen Münch, Andreas Jedlitschka, and Markku Oivo. 2018. Empirical software engineering experts on the use of students and professionals in experiments. *EMSE* 23, 1 (2018), 452–489.
[8] Robert Feldt, Thomas Zimmermann, Gunnar R Bergersen, Davide Falessi, Andreas Jedlitschka, Natalia Juristo, Jürgen Münch, Markku Oivo, Per Runeson, Martin Shepperd, et al. 2018. Four commentaries on the use of students and professionals in empirical software engineering experiments. *EMSE* 23, 6 (2018), 3801–3820.
[9] Juan Antonio Gómez-Gutiérrez, Robert Clarisó, and Jordi Cabot. 2022. A Tool for Debugging Unsatisfiable Integrity Constraints in UML/OCL Class Diagrams. In *BPMDS*. Springer, 267–275.
[10] Robbert Jongeling, Federico Ciccozzi, Jan Carlson, and Antonio Cicchetti. 2022. Consistency Management in Industrial Continuous Model-Based Development Settings: A Reality Check. *SoSym* 21, 4 (aug 2022), 1511–1530.
[11] Amy J Ko, Thomas D LaToza, and Margaret M Burnett. 2015. A practical guide to controlled experiments of software engineering tools with human participants. *EMSE* 20, 1 (2015), 110–141.
[12] Roland Kretschmer, Djamel Eddine Khelladi, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2020. Consistent change propagation within models. *SoSym* (2020), 1–17.
[13] Nuno Macedo, Tiago Jorge, and Alcino Cunha. 2017. A Feature-Based Classification of Model Repair Approaches. *IEEE TSE* 43, 7 (2017), 615–640.
[14] Luciano Marchezan, Wesley K. G. Assuncao, Roland Kretschmer, and Alexander Egyed. 2022. Change-Oriented Repair Propagation. In *ICSSP*. ACM, 82–92.
[15] Luciano Marchezan, Wesley K. G. Assunção, Gabriela K. Michelon, and Alexander Egyed. 2023. Experiment's Online Appendix. https://sites.google.com/view/rrexperiment/home.
[16] Luciano Marchezan, Roland Kretschmer, Wesley KG Assunção, Alexander Reder, and Alexander Egyed. 2022. Generating repairs for inconsistent models. *SoSym* (2022), 1–33.
[17] Rahul Mohanani, Iflaah Salman, Burak Turhan, Pilar Rodríguez, and Paul Ralph. 2020. Cognitive Biases in Software Engineering: A Systematic Mapping Study. *IEEE TSE* 46, 12 (2020), 1318–1339.
[18] Yannic Noller, Ridwan Shariffdeen, Xiang Gao, and Abhik Roychoudhury. 2022. Trust Enhancement Issues in Program Repair. In *ICSE*.
[19] Mie Nørgaard and Kasper Hornbæk. 2006. What Do Usability Evaluators Do in Practice? An Explorative Study of Think-Aloud Testing. In *DIS*. ACM, 209–218.
[20] Manuel Ohrndorf, Christopher Pietsch, Udo Kelter, Lars Grunske, and Timo Kehrer. 2021. History-Based Model Repair Recommendations. *ACM Trans. Softw. Eng. Methodol.* 30, 2, Article 15 (Jan. 2021), 46 pages.
[21] Kai Petersen, Kari Rönkkö, and Claes Wohlin. 2008. The Impact of Time Controlled Reading on Software Inspection Effectiveness and Efficiency: A Controlled Experiment. In *ESEM*. ACM, 139–148.
[22] Alexander Reder and Alexander Egyed. 2012. Computing repair trees for resolving inconsistencies in design models.. In *ASE*, Michael Goedicke, Tim Menzies, and Motoshi Saeki (Eds.). ACM, 220–229.
[23] Alexander Reder and Alexander Egyed. 2013. Determining the Cause of a Design Model Inconsistency. Miner Revision. *TSE* (2013).
[24] Robert Rosenthal and Ralph L Rosnow. 2008. *Essentials of behavioral research: Methods and data analysis*. McGraw-Hill.
[25] Weslley Torres, Mark GJ Van den Brand, and Alexander Serebrenik. 2020. A systematic literature review of cross-domain model consistency checking by model management tools. *SoSym* (2020), 1–20.
[26] Sira Vegas, Cecilia Apa, and Natalia Juristo. 2016. Crossover Designs in Software Engineering Experiments: Benefits and Perils. *IEEE TSE* 42, 2 (2016), 120–135.
[27] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.