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

Double Anonymous

***Abstract*—Repairing design models is a laborious task that requires a considerable amount of time and effort from developers. Repair recommendation (RR) approaches can be used to reduce the effort and improve the quality of the repairs performed. Such approaches have been evaluated in a variety of case studies in terms of scalability, correctness, and minimalism. These evaluations, however, do not investigate how developers perceive the use of RRs and what are the benefits of using RRs. This investigation is important to understand the developers' perspective and plan improvements in RRs approaches to meet developers' preferences and needs. In this work, we report a controlled experiment carried out with 24 developers where they repaired design models in eight different tasks, with RRs being provided in half of the tasks. We compare the results of the tasks where RRs were provided with results from tasks where no RRs were given. The findings indicate that developers benefit from RRs by improving the effectiveness and efficiency when repairing design models in more complex tasks. The results also evidence that RRs do 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 alternatives to fix the inconsistency that were not considered without RRs.**

***Index Terms*—Controlled Experiment, Consistency Checking, Repair Recommendations**

I. Introduction

Consistency checking can be used to repair design models, improving the models' quality and reducing the chance of requirements not being met [1], [2]. Repairing models, however, is an error-prone and laborious task requiring a considerable amount of time and effort from developers [3], [4]. Hence, repair recommendation (RR) approaches are applied to maintain quality and reduce the time and effort required for the repair, mainly on UML models [5], [6].

Approaches proposing the use of RRs have been evaluated in a variety of case studies in terms of scalability, correctness, and minimalism [7]–[13]. In this context, user studies are also relevant to analyze the implications of applying an RRs strategy from the perspective of its end-users, i.e., developers [14], [15]. Most studies, however, do not focus their evaluations on the perspective of what developers think and how they may benefit from the provision of RRs. Furthermore, as developers have different experiences and preferences, ranking or automatic executing RRs to fix inconsistencies may change the models in a way not desired by the developer. Hence, it is important to understand the developers' preferences when repairing models.

In this paper, we report an experiment to investigate if RRs bring benefits to developers in terms of effectiveness and efficiency when repairing design models. Moreover, we also analyze the RRs selected and created by the developers to understand their preferences for different tasks. The goal of our study is described 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?

Our experiment was conducted by analyzing the results of developers when repairing inconsistent models with and without the provision of RRs. The sample was composed of 24 participants with varied software development experience, ranging from academic only to more than five years of experience in the industry. To answer RQ1, we measure inconsistencies fixed (effectiveness) and the completion time required (efficiency). 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.

The results of the experiment show that the use of RRs can benefit developers by improving their effectiveness by 37.36% and efficiency by 29.81% (RQ1). These findings evidence the importance of RRs approaches and how they can benefit developers. Furthermore, the perceived difficulty and confidence of developers are not impacted by the provision or lack of RRs, with 88% of the participants saying that it is better to use RRs than not use them (RQ2). This result demonstrates that RRs approaches can be applied without drawbacks from the developers' perspective. We also observed that developers do not have an ideal RR for an 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 the drawbacks of using automatic repairs in models.

This work is organized as follows: Section II describes the background of repair generation as well as related work. Section III presents the design and the threats to the validity of the controlled experiment. The results and discussion of the RQs are presented in Section IV. Lastly, Section V presents our conclusions and future work.

II. Background and Related Work

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

A. Repair Recommendations for Design Models

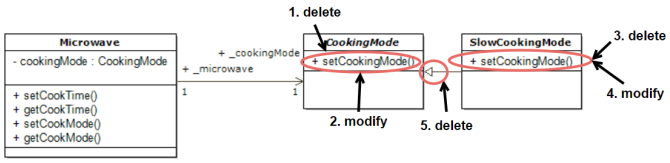
Consistency checking is used to analyze design models and identify inconsistencies to be repaired. In this context, we consider a model as consisting of model elements that contain properties, e.g., a UML model. A property can be of a primitive type, e.g., Integer or String, or a reference to other model elements. 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 defined for a context that must be fulfilled by a model element. This condition 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 the inconsistencies are found, repair recommendations (RR) are generated to fix them. In this sense, a RR defines a change of a model element property that resolves an inconsistency. A RR identifies the operator, the model element, the model element property, and, optionally, a value to change the model element property. 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. An abstract RR is an RR where no value can be calculated. 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 *del*. Formalizing, this RR would be: $(del, CookingMode.operations, setCookingMode)$. We adapted approaches found in the literature [4], [12], [16] to transform this formal representation to 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.

We implemented an approach that analyzes model inconsistencies and generates the RRs used in the experiment. The approach used is implemented based on repair generation approaches from the literature [4], [12], [16]. There are different rationales for selecting these approaches to be used as the basis for implementing the one applied in our experiment. For instance, their robustness, since these approaches have been

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**

Fig. 1. Example of task with Repair Recommendations

evaluated in large-scale case studies 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. Details about how RRs are generated are described in the following.

Figure 1 describes a CR that iterates over all operations of a class.[1] 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 the RRs. In this case, the generator function is applied to a universal quantifier (*forAll*) as described by the OCL definition of the CR.[1] This generator function suggests both modifying or deleting the inconsistent model elements [4], [12]. Additionally, it suggests deleting the inheritance association, since this association is responsible for the set of operations. This results in the five RRs 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. Notice that for the RRs shown in Figure 1, while three of the RRs are concrete (*1. Delete*, *3. Delete*, and *5. Delete*) the other two (*2. Modify* and *4. Modify*) are abstract since they suggest modifying the signature without saying how. In these cases, a developer is required to define a value for the RRs.

[1]All artifacts of the experiment are available at our online appendix [17]

Only one RR is required to fix any given inconsistency. The approach treats every option equally, thus, they are not ranked in any particular order. Rather, they are randomly ordered, as displayed in Figure 1. Hence, the rank column remains empty, and it is the developer that should select and rank the RRs.

B. Related Work

Several approaches proposed the use of RRs [2]. These approaches use different strategies for generating repairs, such as relying on OCL rules and generator functions [4], [12], [16], [18], analysing the history of changes [10], [19], [20], and applying learning algorithms [13], [21], [22]. 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. Furthermore, there is a lack of studies analyzing the implications of providing RRs to developers, such as benefits, challenges, and preferences. Thus, we designed our experiment with the goal to collect and analyze these implications.

When designing our experiment, we applied the guidelines for experimentation with human participants in software engineering [23]–[25]. Furthermore, we have to consider how other experiments with human participants were designed [26], [27]. Petersen et al. [28] report an experiment to analyze the impact of time-controlled reading inspection regarding effectiveness and efficiency. When formulating two metrics for RQ1, namely, effectiveness and efficiency, we considered Petersen et al.'s formulas to calculate the scores for each task of the experiment. When designing the methods for analyzing the results of our experiment, we considered studies that had similar protocols. For instance, Dzidek et al. [29] 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 use and the lack of RRs when repairing design models. Moreover, their experiment was also composed of quantitative and qualitative RQs. Thus, we also have quantitative and qualitative RQs. Analyzing works that deal with questionnaires is also important when formulating the questions, metrics, and analyzing results from those questionnaires [30], [31].

Since the experiment's sample was composed of developers that are M.Sc./Ph.D. students, we have to consider guidelines regarding students as participants in experiments [32], [33]. Fallessi et al. [32], 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 Fallessi et al., conducting experiments with students does not have a lower relevance or lower interest than experiments with professionals. In the same context, Feldt et al. [33] 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. Details about the experiment design, including the sample selection, are described in the next section.

III. Experiment Design

In this section, we present the experiment design[1] defined based on the guidelines presented by Wohlin et al. [23] and Ko et al. [24].

A. Experiment Definition and Planning

The main goal of our experiment is to analyze if developers benefit from the provision of RRs and 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.

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 [34], [35], as well as being commonly used in controlled experiments [36]–[41].

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: $\mathbb{T}$: Time spent on the tasks; $\mathbb{I}$: number of inconsistencies per task; $\mathbb{IF}$: number of inconsistencies fixed per task. Hence, we attribute different $\mathbb{IF}$ scores as an inconsistency could be fixed (score 1), partially fixed (score 0.5), or not fixed (score 0); $\mathbb{RRR}ank$: the ranking of repair recommendations given by the participants in RR tasks; $\mathbb{RC}$: repairs created by participants in WR tasks. In addition, we use four qualitative metrics, presented in Table I: 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 [28], namely, effectiveness (EFT) and efficiency (EFY). Effectiveness 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. Efficiency is the degree to which participants repair a model inconsistency, considering the time required (Equation 2).

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

These two metrics are used to quantify the possible benefits regarding effectiveness and efficiency. We also complement the analysis of the results by extracting themes from open-ended questions present in the feedback questionnaires. These questions included asking participants about the challenges faced (see $\mathbb{CF}$ in Table I) 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 [42], [43], 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 participants' effectiveness and efficiency when performing repairs in inconsistent models with and without RRs. With these results, we analyze if the provision of RRs brings them benefits regarding these two metrics. **Method:** we answer this RQ by testing hypotheses for the EFT and EFY metrics. For the EFT metric, the null hypothesis H_0 states that RR and WR have the same effectiveness. The alternative hypothesis H_a states that the effectiveness is higher for RR.

$$H_{0EFT} := EFT(RR) = EFT(WR)$$
$$H_{aEFT} := EFT(RR) > EFT(WR)$$

For the EFY metric, the null hypothesis H_0 states that RR and WR have the same efficiency. The alternative hypothesis H_a states that the efficiency is higher for RR.

$$H_{0EFY} := EFY(RR) = EFY(WR)$$
$$H_{aEFY} := EFY(RR) > EFY(WR)$$

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. These results complement RQ1's results since EFT and EFY are related to participants' perceived difficulty and confidence level, e.g., a task considered easy by participants may have higher EFT and EFY results. **Method:** we consider four questions from the feedback questionnaires given to participants, namely, $\mathbb{TD}$, $\mathbb{CL}$, $\mathbb{OD}$, and $\mathbb{PA}$, shown in Table I. We also consider the two metrics used for the hypothesis testing performed for RQ1 when analyzing the correlation between the perceived difficulty and the EFT and EFY 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 a 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 I
QUESTIONS FROM THE EXPERIMENT QUESTIONNAIRES

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

2) Sample Selection: We aimed at having developers with both academic and industrial experience, obtaining a group of participants with varying levels of experience. We invited participants for the experiment using convenience sampling, i.e., participants selected were easily accessible [23]. 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 [32]. 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 III-C.

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 II, 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. As shown in the results, 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) Experimental Package: The experimental package used was designed following a strategy closely related to the strategy for mitigating threats to validity (see Section III-C). The package is composed of the following artifacts:

Design models: the design models used are defined in UML and are part of a dataset of both industry and academic models of different domains.[1] This dataset has been used for evaluating repair generation approaches [12], [16], [44]. We

TABLE II
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

used six different UML models, composed of class, sequence, and state machine diagrams. The selection of models was performed considering the mitigation strategy for TtV4, TtV7, TtV10, and TtV11 (Section III-C).

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 responsible for carrying out the session. The definition of the guidelines was based on the mitigation from TtV5.

Tasks' document: the tasks of the experiment were given to participants as a printed document. When designing the tasks, we had to consider the diagrams to be used and their inconsistencies. 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 III 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 III-C, TtV7). For each main task, we used a unique CR, six CRs in total, to identify an inconsistency in the diagram. For the G4 tasks, we reapplied two CRs already used in previous tasks. 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 presented were varied, for some tasks we introduced the inconsistency, i.e., by deleting or modifying an element from the diagram. We grouped tasks in pairs based on the diagram types being used, e.g., G3 is composed of TSeq and TSta which are the 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 III-B). As tasks of the same group were always given together, we ensure that the most similar tasks were closer. The tasks' document was defined based on the mitigation strategies for TtV3, TtV5, TtV6, TtV7, TtV8, and TtV9 detailed in Section III-C.

TABLE III
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

Repair recommendations: RRs were generated for all tasks. Each participant, however, received RRs for only four tasks, having to perform the other tasks without RRs. Hence, for each group of tasks, one task was presented with RRs and the other task without recommendations (WR). For instance, for Participant 1, if TCex had RRs, then TCac did not have them. For Participant 2, however, TCex did not have RRs and TCac did. We ensured that no matter the order of the tasks given to each participant, the tasks would alternate between RR and WR. When a task contained RR, 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 could rank RRs by 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 repairs, if they believed that the RRs given were not all the possible ones. This document was designed considering the mitigation strategy for threats TtV5 and TtV6 as described in Section III-C.

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 I. After the completion of all tasks, participants were given an overall feedback questionnaire with questions $\mathbb{OD}$, $\mathbb{PA}$, and $\mathbb{CF}$ from Table I among others.[1] These questionnaires were designed considering the mitigation strategy for threats TtV3 and TtV7 (Section III-C).

B. 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 a debriefing about the experiment's main goal, without mentioning RQs or hypotheses. One experimenter read aloud the guidelines document and asked 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 always write down the current time before and after each task.

After all the participants had finished reading the guidelines, the tasks' 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 tasks' documents printed, one for each participant. All documents had the same eight tasks (see Table III) 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 III-C). 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.

C. Threats to Validity

In this section, we discuss internal, external, and conclusion threats to validity (TtV) and how we mitigated them based on the literature [23], [45].

1) Construct Validity: The first threat is using a convenience sample, as it can bias the results based on the participants' experience (TtV1). *Mitigation*: we applied a questionnaire with the R^3 characterization mechanism [32] 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 [24], [46]. *Mitigation*: we collected participants' demographic data through an online form, one week before the experiment. Also, the options provided as possible answers to close-ended questions can be a threat as they may confuse or unintentionally bias participants' answers (TtV3) [46]. *Mitigation*: 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.

2) Internal Validity: For internal validity, we consider the experimental package. Firstly, the models selected may bias the results of the RQs (TtV4). *Mitigation*: the models from the dataset were already evaluated and used in other studies [12], [16], [44], thus they were 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 III. Considering the operation of the experiment, participants not reading and not understanding the instructions can also threaten the results (TtV5). *Mitigation*: participants were asked to silently read the guidelines provided while an experimenter read these same guidelines aloud. Participants had the opportunity to ask questions about the guidelines. The guidelines also contained two illustrative tasks that simulated the two types of tasks (RR and WR). Another threat is the tasks' order and structure impacting the results (TtV6) due to the learning effect [24], [47]. *Mitigation*: we structured the experiment tasks using a counterbalancing strategy [24], [48]. 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.[2]

Another threat is related to task difficulty and completion time (TtV7) that could impact the results as it can create fatigue in the participants [24]. *Mitigation*: we conducted 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 participants to ask for feedback related to the pilot study. Based on the feedback given 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). *Mitigation*: as the repair approach used generates every possible RR for each inconsistency, we checked if the repairs created by participants in WR tasks were part of the generated repairs 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 EFY (TtV9). *Mitigation*: we asked participants to only start the tasks after the debriefing and consent form agreement. 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 related to each task. To keep the time consistent, we asked participants to use the time displayed in a digital clock displayed on the beamer.

3) External Validity: Considering external validity, the impossibility to generalize the results to practical cases due to the participants of the experiment being students is a threat (TtV10). *Mitigation*: we only invited participants that already concluded their bachelor's degree. As shown in Table II, 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

[2]The definition of the task order is presented in our online appendix [17].

industrial practice (TtV11). *Mitigation*: the models used in this experiment were taken from real systems, but 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.

IV. Results and Discussion

In this section, we present the results and answer the RQs.

RQ1. Do developers benefit from recommendations when repairing inconsistent design models? Table IV describes the results of the EFT and EFY metrics per task, distinguished by task type (RR and WR). Considering the effectiveness result, column *EFT*, the use of RR always led to 1 EFT (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 EFT 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 EFT 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 EFT scores for these tasks were among the worst. We observed that the reason for these lower EFT 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 tasks that obtained the best EFT results, i.e., tasks TCac, TMic, TCdr, TSeq, and TSta. Thus, the results regarding EFT indicate that *participants can benefit from RRs by improving the effectiveness when repairing more complex inconsistencies while keeping the similar effectiveness for simpler inconsistencies*.

Figure 2 illustrates the results related to the time ($\mathbb{T}$) required to complete each task. The average time is relatively 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 RR 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, participants' 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 IV, five tasks (TCex, TMic, TCdr, TCno, and TSup) were performed with more efficiency when RRs were given. The other three tasks (TCac, TSeq, and TSta) obtained higher EFY results when no RRs were given. Thus, *RRs benefit participants by improving the efficiency when repairing more complex inconsistencies*.

TABLE IV
Results of EFT and EFY per task and overall

Task	Type	$\mathbb{I}$	$\mathbb{IF}$ (avg.)	EFT	$\mathbb{T}$ (avg.)	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		

The results of the hypotheses testing, shown at the bottom of Table IV, compare the mean, median, and standard deviation of the metrics EFT and EFY. 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 EFT or EFY, whereas a negative value means that RR is worse. Considering EFT, the use of RR showed to be 37.63% better than WR. To test the statistical significance of the results, we performed a two-tailed distribution, paired *t*-test. The EFT result is statistically significant, since the *t*-test resulted in a p-value of 0.04 (column *p-value*). The result considering the EFY metric shows that the use of RR is 29.81% better than WR. This result, however, is not statistically significant (p-value = 0.17).

The individual EFY result per task is the main reason why the p-value for EFY was higher. For example, the results for task TCex were the worst considering WR tasks. This was also the task where participants spent the most time on average (5.58 minutes, as shown in Table IV). Since the time is used to generate the EFY results, this leads to TCex WR having the lowest EFY score (3.14). We argue that the main reason for that is the difficulty of the task. According to 42% of the participants, task TCex WR was *difficult*, which represents the

number four in a 1-5 Likert scale.[3] A parallel can be traced to TSta WR which obtained the highest EFY result (23.26). The majority of participants (50%) perceived TSta WR to be *easy*. This result is also reflected in the time spent on TSta WR, as participants spent less time on average (2.59 minutes). To summarize the results of RQ1, we conclude that:

> ***Answering RQ1:*** *The use of RRs benefits participants improving effectiveness and efficiency when repairing complex inconsistencies, being 37.63% more effective (p = 0.04) and 29.81% more efficient overall (p = 0.17). Furthermore, RRs can keep the completion time required more stable.*

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, leading to a longer time. We argue, however, that even in more simple tasks, RRs bring benefits since they can provide alternatives that initially may not be considered by participants. Further discussion about this argument is given in the discussion of RQ3's results.

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 repairs required*, similar to RQ1's results.

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.[3] An exception is TSeq, where the majority of participants (42%) were *fairly confident* without RRs, representing the number four in the 1-5 Likert scale. Moreover, for RR tasks, the majority of participants (50%) were *somewhat confident*, representing the number three on the 1-5 Likert 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 both 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. In this sense, improving the process of repairing models is important to formalize the steps, artifacts, roles, and workflow of how inconsistencies should be identified and repaired in the software-engineering life cycle. This conclusion, however, is only a possible implication found by the results, since this topic is out of the scope of this work.

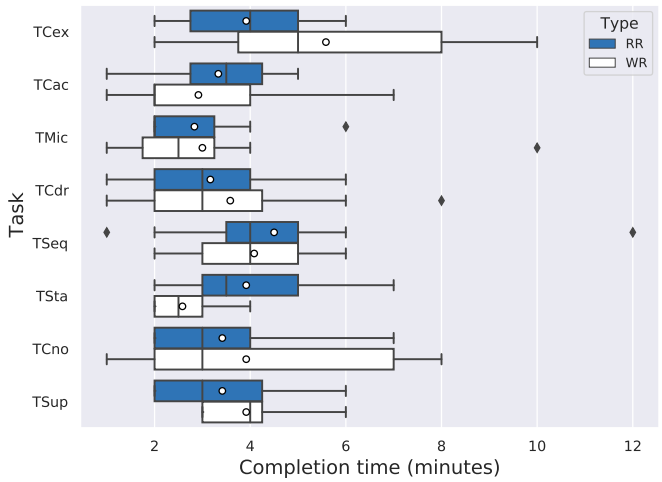


Fig. 2. Result of time (in minutes) spent on tasks

We also compare the difficulty and confidence results with the EFT and EFY scores from Table IV. These results are 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 50% EFT and below 10 EFY. Furthermore, tasks TCac, TMic, and TCdr are considered either *easy* or *very easy* and stayed between 90% and 100% EFT and between 10 and 20 EFY. Similarly, the confidence level may also be associated with the EFT and EFY results. In some RR tasks (TCex, TCno), however, although participants were always able to fix the inconsistency (EFT = 100%), they still do not feel confident about it. This result indicates 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*. Concerning the participants' acceptance about the use of RRs ($\mathbb{PA}$), the majority of participants (50%) answered that using

[3]Results of the questionnaires are available at our online appendix [17]

RRs when fixing inconsistent models is *much better* than not using them.[4] The option *somewhat better* was selected by 38%, with *about the same* being 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 fixing models. We can summarize the results of RQ2 as:

> ***Answering RQ2:*** *The provision or the lack of RRs does not impact participants' perceived difficulty and confidence, although, RRs lead to a higher EFT and EFY. Participants, however, feel that using RRs is better than not using them.*

Furthermore, the results related to developers' confidence are 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 do not expect developers' feedback. These approaches produce correct repairs, however, those repairs may not be *correct* or the *ideal* alternative in the eyes of the developer, which is further discussed in RQ3.

RQ3. Do developers have preferred recommendations when repairing inconsistent design models? Table V shows the results for the RR ranking ($\mathbb{RRR}$) for each RR task. First, we analyze the RRs that were selected as the first option in each task. Column *Ranked 1st #* shows the number of times that a 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 V shows results for the overall ranking of RRs per task. The average ranking (displayed in the column *Avg.*) shows that for most tasks, at least one RR stayed close to the bottom rank. This means that such a 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 a 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 V, which describes the worst ranking given for each RR. These results indicate that, *depending on the context, any RR may be considered not applicable*. 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*. 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 a 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. However, these potential implications should be further

TABLE V
RESULTS OF RRS RANKING

Task	RR	Ranked 1st* #	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%

[4]Results of all questions are available at our online appendix [17]

TABLE VI
REPAIRS CREATED (RC) IN WR TASKS

Task	RC	#	%	Related RR	Task	RC	#	%	Related 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

explored in studies focusing on side effects and RRs rationale, which are is of the scope of this work.

Considering the best ranking (column *Best*) for most tasks (TCex, TCac, TMic, TCdr, TSeq, TCno, and TSup) 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 VI describes the repairs created (RC) by the participants in WR tasks. We analyzed all WR tasks and counted the number of times (column #) that the same RC was performed by a participant. While performing this analysis, we only considered RC that fixed the inconsistencies in the task. Hence, tasks where participants were not able to fix the inconsistency, were not considered since the proposed changes in the model are not repairs. Tasks TCex and TCno had only one RC each, these are the same tasks with the lowest EFT and EFY 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 (for seven participants) only partially fixed the inconsistency in the task. For both tasks, TCex and TCno, the number of participants that were 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 VI, 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 V and VI 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 comments of two participants, mentioning that *some RRs alternatives were not considered when RRs were not given*.

The column *Related RR* in Table VI 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 III-C), 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*. We can summarize the results for RQ3 as:

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.*

V. CONCLUSION AND FUTURE WORK

We identified the lack of a user study to evaluate the benefits of RRs when repairing inconsistent models. 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 repairs required are complex. We also observed that the perceived difficulty and confidence of developers are not impacted by the provision of RR. Most developers, however, agreed that using RRs is better than not using them. Furthermore, we concluded that not all developers choose the same RR, but rather, have varied preferences. We also observed that the provision of RRs leads to developers considering more alternatives for fixing an inconsistency. These findings can be used by researchers to understand what are developers' preferences when applying RRs approaches. Addressing these preferences can lead to improvement of presentation and feedback of RRs approaches. Furthermore, we argue that the investigation of the benefits and challenges of side effects and change propagation of RRs from the developers' perspective and the formalization of a repair process for design models are open research opportunities that should be explored.

VI. DATA AVAILABILITY

The experiment's artifacts are available in an online appendix [17].

REFERENCES

[1] W. Torres, M. G. Van den Brand, and A. Serebrenik, "A systematic literature review of cross-domain model consistency checking by model management tools," Software and Systems Modeling, pp. 1–20, 2020.

[2] N. Macedo, T. Jorge, and A. Cunha, "A feature-based classification of model repair approaches," IEEE Transactions on Software Engineering, vol. 43, no. 7, pp. 615–640, 2017.

[3] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," IEEE Transactions on Software Engineering, vol. 37, no. 2, pp. 188–204, 2011.

[4] A. Reder and A. Egyed, "Computing repair trees for resolving inconsistencies in design models." in ASE, M. Goedicke, T. Menzies, and M. Saeki, Eds. ACM, 2012, pp. 220–229. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351707

[5] F. J. Lucas, F. Molina, and A. Toval, "A systematic review of uml model consistency management," Information and Software Technology, vol. 51, no. 12, pp. 1631–1645, 2009, quality of UML Models. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584909000433

[6] R. S. Bashir, S. P. Lee, S. U. R. Khan, V. Chang, and S. Farid, "Uml models consistency management: Guidelines for software quality manager," International Journal of Information Management, vol. 36, no. 6, Part A, pp. 883–899, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401216303425

[7] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein, "xlinkit: a consistency checking and smart link generation service." ACM Trans. Internet Techn., vol. 2, no. 2, pp. 151–185, 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514186

[8] R. Kretschmer, D. E. Khelladi, A. Demuth, R. E. Lopez-Herrejon, and A. Egyed, "From Abstract to Concrete Repairs of Model Inconsistencies: An Automated Approach," in 2017 24th Asia-Pacific Software Engineering Conference (APSEC), 2017, pp. 456–465.

[9] D. E. Khelladi, R. Kretschmer, and A. Egyed, "Detecting and Exploring Side Effects When Repairing Model Inconsistencies," in Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering, ser. SLE 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 113–126. [Online]. Available: https://doi.org/10.1145/3357766.3359546

[10] M. Ohrndorf, C. Pietsch, U. Kelter, and T. Kehrer, "ReVision: A Tool for History-Based Model Repair Recommendations," in Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 105–108. [Online]. Available: https://doi.org/10.1145/3183440.3183498

[11] A. Barriga, A. Rutle, and R. Heldal, "Personalized and automatic model repairing using reinforcement learning," in 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2019, pp. 175–181.

[12] L. Marchezan, R. Kretschmer, W. K. Assunção, A. Reder, and A. Egyed, "Generating repairs for inconsistent models," Software and Systems Modeling, pp. 1–33, 2022.

[13] A. Barriga, R. Heldal, A. Rutle, and L. Iovino, "Parmorel: a framework for customizable model repair," Software and Systems Modeling, pp. 1–24, 2022.

[14] D. Sjoeberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. Rekdal, "A survey of controlled experiments in software engineering," IEEE Transactions on Software Engineering, vol. 31, no. 9, pp. 733–753, 2005.

[15] M. Nørgaard and K. Hornbæk, "What do usability evaluators do in practice? an explorative study of think-aloud testing," in Proceedings of the 6th Conference on Designing Interactive Systems, ser. DIS '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 209–218. [Online]. Available: https://doi.org/10.1145/1142405.1142439

[16] R. Kretschmer, D. E. Khelladi, R. E. Lopez-Herrejon, and A. Egyed, "Consistent change propagation within models," Software and Systems Modeling, pp. 1–17, 2020.

[17] "Experiment's online appendix," https://sites.google.com/view/rrexperiment/home.

[18] J. A. Gómez-Gutiérrez, R. Clarisó, and J. Cabot, "A tool for debugging unsatisfiable integrity constraints in uml/ocl class diagrams," in International Conference on Business Process Modeling, Development and Support, International Conference on Evaluation and Modeling Methods for Systems Analysis and Development. Springer, 2022, pp. 267–275.

[19] M. Ohrndorf, C. Pietsch, U. Kelter, L. Grunske, and T. Kehrer, "History-Based Model Repair Recommendations," ACM Trans. Softw. Eng. Methodol., vol. 30, no. 2, Jan. 2021. [Online]. Available: https://doi.org/10.1145/3419017

[20] L. Marchezan, W. K. G. Assuncao, R. Kretschmer, and A. Egyed, "Change-oriented repair propagation," in Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering, ser. ICSSP'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 82–92. [Online]. Available: https://doi.org/10.1145/3529320.3529330

[21] A. Barriga, R. Heldal, L. Iovino, M. Marthinsen, and A. Rutle, "An extensible framework for customizable model repair," in Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 24–34. [Online]. Available: https://doi.org/10.1145/3365438.3410957

[22] A. Barriga, L. Mandow, J. L. P. de la Cruz, A. Rutle, R. Heldal, and L. Iovino, "A comparative study of reinforcement learning techniques to repair models," in Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3417990.3421395

[23] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in software engineering. Springer Science & Business Media, 2012.

[24] A. J. Ko, T. D. LaToza, and M. M. Burnett, "A practical guide to controlled experiments of software engineering tools with human participants," Empirical Software Engineering, vol. 20, no. 1, pp. 110–141, 2015.

[25] S. Easterbrook, "Empirical research methods for software engineering," in Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering, ser. ASE '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 574. [Online]. Available: https://doi.org/10.1145/1321631.1321749

[26] E. P. Enoiu, A. Cauevic, D. Sundmark, and P. Pettersson, "A controlled experiment in testing of safety-critical embedded software," in 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), 2016, pp. 1–11.

[27] I. Karac, B. Turhan, and N. Juristo, "A controlled experiment with novice developers on the impact of task description granularity on software quality in test-driven development," IEEE Transactions on Software Engineering, vol. 47, no. 7, pp. 1315–1330, 2021.

[28] K. Petersen, K. Rönkkö, and C. Wohlin, "The impact of time controlled reading on software inspection effectiveness and efficiency: A controlled experiment," in Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ser. ESEM '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 139–148. [Online]. Available: https://doi.org/10.1145/1414004.1414029

[29] W. J. Dzidek, E. Arisholm, and L. C. Briand, "A realistic empirical evaluation of the costs and benefits of uml in software maintenance," IEEE Transactions on Software Engineering, vol. 34, no. 3, pp. 407–432, 2008.

[30] M. Lillack, S. Stanciulescu, W. Hedman, T. Berger, and A. Wasowski, "Intention-based integration of software variants," in 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 831–842.

[31] Y. Noller, R. Shariffdeen, X. Gao, and A. Roychoudhury, "Trust enhancement issues in program repair," in Proceedings of the ACM/IEEE 44th International Conference on Software Engineering, 2022.

[32] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, "Empirical software engineering experts on the use of students and professionals in experiments," Empirical Software Engineering, vol. 23, no. 1, pp. 452–489, 2018.

[33] R. Feldt, T. Zimmermann, G. R. Bergersen, D. Falessi, A. Jedlitschka, N. Juristo, J. Münch, M. Oivo, P. Runeson, M. Shepperd et al., "Four commentaries on the use of students and professionals in empirical software engineering experiments," Empirical Software Engineering, vol. 23, no. 6, pp. 3801–3820, 2018.

[34] J. Hutchinson, J. Whittle, and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," Science of Computer Programming, vol. 89, pp. 144–161, 2014, special issue on Success Stories in Model Driven Engineering. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642313000786

[35] F. Ciccozzi, I. Malavolta, and B. Selic, "Execution of uml models: a systematic review of research and practice," Software & Systems Modeling, vol. 18, no. 3, pp. 2313–2360, 2019.

[36] R. Conradi, P. Mohagheghi, T. Arif, L. C. Hegde, G. A. Bunde, and A. Pedersen, "Object-oriented reading techniques for inspection of uml models–an industrial experiment," in European Conference on Object-Oriented Programming. Springer, 2003, pp. 483–500.

[37] C. Bunse, "Using patterns for the refinement and translationof uml models: A controlled experiment," Empirical Software Engineering, vol. 11, no. 2, pp. 227–267, 2006.

[38] A. Nugroho, "Level of detail in uml models and its impact on model comprehension: A controlled experiment," Information and Software Technology, vol. 51, no. 12, pp. 1670–1685, 2009, quality of UML Models. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584909000408

[39] A. M. Fernández-Sáez, M. Genero, M. R. Chaudron, D. Caivano, and I. Ramos, "Are forward designed or reverse-engineered uml diagrams more helpful for code maintenance?: A family of experiments," Information and Software Technology, vol. 57, pp. 644–663, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914001311

[40] G. Scanniello, C. Gravino, M. Genero, J. A. Cruz-Lemus, G. Tortora, M. Risi, and G. Dodero, "Do software models based on the uml aid in source-code comprehensibility? aggregating evidence from 12 controlled experiments," Empirical software engineering, vol. 23, no. 5, pp. 2695–2733, 2018.

[41] Á. Domingo, J. Echeverría, Ó. Pastor, and C. Cetina, "Comparing uml-based and dsl-based modeling from subjective and objective perspectives," in International Conference on Advanced Information Systems Engineering. Springer, 2021, pp. 483–498.

[42] D. S. Cruzes and T. Dybå, "Synthesizing evidence in software engineering research," in Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ser. ESEM '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1852786.1852788

[43] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in 2011 International Symposium on Empirical Software Engineering and Measurement, 2011, pp. 275–284.

[44] A. Reder and A. Egyed, "Determining the Cause of a Design Model Inconsistency," Transaction on Software Engineering (TSE), 2013.

[45] J. Siegmund, N. Siegmund, and S. Apel, "Views on internal and external validity in empirical software engineering," in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, 2015, pp. 9–19.

[46] R. Mohanani, I. Salman, B. Turhan, P. Rodríguez, and P. Ralph, "Cognitive biases in software engineering: A systematic mapping study," IEEE Transactions on Software Engineering, vol. 46, no. 12, pp. 1318–1339, 2020.

[47] R. Rosenthal and R. L. Rosnow, Essentials of behavioral research: Methods and data analysis. McGraw-Hill, 2008.

[48] S. Vegas, C. Apa, and N. Juristo, "Crossover designs in software engineering experiments: Benefits and perils," IEEE Transactions on Software Engineering, vol. 42, no. 2, pp. 120–135, 2016.