Journal of Object Technology
Published by AITO — Association Internationale pour les Technologies Objets
http://www.jot.fm/

An Empirical Study on the Impact of Inconsistency Feedback during Model and Code Co-changing

Georgios Kanakis[a] Djamel Eddine Khelladi[b] Stefan Fischer[a]
Michael Tröls[a] Alexander Egyed[a]

a. Johannes Kepler University, Institute of Software Systems Engineering, 4040 Linz, Austria

b. University Rennes 1, CNRS UMR6074, IRISA, Rennes, France

Abstract Model and code co-changing is about the coordinated modification of models and code during evolution. Intermittent inconsistencies are a common occurrence during co-changing. A partial co-change is the period in which the developer changed, say, the model but has not yet propagated the change to the code. Inconsistency feedback can be provided to developers for helping them to complete partial co-changes. However, there is no evidence whether such inconsistency feedback is useful to developers. To investigate this problem, we conducted a controlled experiment with 36 subjects who were required to complete ten partially completed change tasks between models and code of two non-trivial systems. The tasks were of different levels of complexity depending on how many model diagrams they affected. All subjects had to work on all change tasks but sometimes with and sometimes without inconsistency feedback. We then measured differences between task effort and correctness. We found that when subjects were given inconsistency feedback during tasks, they were 268% more likely to complete the co-change correctly compared to when they were not given inconsistency feedback. We also found that when subjects were not given inconsistency feedback, they nearly always failed in completing co-change tasks with high complexity where the partially completed changes were spread across different diagrams in the model. These findings suggest that inconsistency feedback (i.e. detection and repair) should form an integral part of co-changing, regardless of whether the code or the model changes first. Furthermore, these findings suggest that merely having access to changes (as with the given partially completed changes) is insufficient for effective co-changing.

Keywords Model Driven Engineering, Consistency Feedback, Change Propagation, Models, Code

1 Introduction

Model-Driven Engineering (MDE) has shown to be effective in the development and maintenance of large scale and embedded systems [BHH05, BSL99]. Models are used in all development stages, from specifying the customers' requirements, design, all the way to source code. The benefits range from increased productivity to reduced time to market [DCF+16]. These benefits, however, hinge on the assumption that model and code remain consistent over time which is often not the case. Changes in models and code cause inconsistencies and subsequent errors if these inconsistencies are not recognized in a timely manner [Egy11]. Any conclusions or automation based on them cannot be trusted for as long as these inconsistencies remain. Inconsistencies are better detected with today's consistency checking mechanisms [KK99]. However, literature tends to treat inconsistencies as errors - typically as errors within models (and their representations). Indeed, there is very little discussion about inconsistencies between model and code due to evolutionary changes.

That is, model and code inconsistencies are often the result of evolutionary changes: changes where the model might have been evolved in response to changing requirements or defects, but the code has not been updated to reflect those changes. Here, inconsistencies imply that either model, code or both are out of date. Consequently, inconsistencies between model and code are cues about the incomplete propagation of changes from model to code or code to model. These inconsistencies occur intermittently - after, say, the engineer changed the model but before the engineer propagated the change to the code. The questions we are interested in here are: do engineers benefit from inconsistency feedback during model and code evolution? Or more concretely: does this inconsistency feedback benefit effort (i.e., save time) or does it improve quality (i.e., result in more complete change propagations)? Or both? The goal of this paper is to explore these questions. To examine the usefulness of inconsistency feedback during model and code co-change, we performed a controlled experiment with 36 subjects - all bachelor level students at the Johannes Kepler University but with average professional work experience of almost 2 years (many students do work part-time). The subjects were confronted with model or source code changes and they were then asked to complete those changes (i.e., to repair the inconsistencies that had been caused by those changes). The key distinguishing factor was that subjects were sometimes given inconsistency feedback and other times not. This way we could measure whether the availability of inconsistency feedback was beneficial to the subjects while they repaired the inconsistencies. Specifically, we measured their effort and their correct/complete propagation.

To ensure wider applicability, we asked each subject to work on ten change request tasks covering two, non-trivial software systems. Five tasks were about changes to an open source monopoly game called "Matador" and the other five tasks were about a proprietary calendar application called "Calendarium". Each task could come in two possible change directions: with the model-to-code change direction, subjects were presented with model changes and were asked to update the code (i.e., propagate changes from model to code). With the code-to-model change direction, subjects were asked to update the model in response to code changes (i.e., propagate changes from code to model). The change direction, task order, and availability of inconsistency feedback was randomized for each subject. Hence, every subject was asked to perform some change propagations with and some without inconsistency feedback; some from model to code and some from code to model; and all tasks in a different order.

Out of the possible 360 tasks (if all 36 subjects had managed to complete all ten change requests), we obtained 281 completed solutions - 7.9/10 completed tasks per subject on average. Our findings showed that subjects working with inconsistency feedback solved 268% more correct tasks than those without inconsistency feedback (59 correct tasks with consistency feedback as opposed to 22 tasks without feedback). We found that the correctness improved regardless of the type of project or the direction of the change. Hence, this benefit is solely the result of the availability of inconsistency feedback. Inconsistency feedback thus helps engineers to correctly propagate changes between model and code.

Surprisingly, the findings seemed to show little difference in effort for subjects with and without inconsistency feedback. A deeper analysis revealed that, when deprived of inconsistency feedback, subjects mostly failed to solve certain tasks entirely. We found that this applied particularly to tasks with higher complexity where the changes affected multiple diagrams of a model (e.g., engineers had to understand a class and a sequence diagram change together to understand how to co-change the code).

Our results imply that inconsistency feedback is an essential complement for change propagation between model and code. Merely understanding model changes is not sufficient to understand how to co-change the code and vice versa. Given the strong correctness benefit, we argue that the use of inconsistency feedback should be common practice not just for error detection but also be change propagations/co-evolution given that there is almost no cost involved in using consistency checking mechanisms (e.g., Nentwich et al [NEFE03], Egyed [Egy11], Musuvathi et al [MPC$^+$02]) and given that many commonly usable consistency rules are available [TLG14]. We believe that there are likely subsequent benefits that stem from the quicker availability of inconsistency feedback for more correct models or code which we did not measure since it was out of the scope of this paper. Given that we chose two reasonably complex projects, ten diversely complex change request tasks, two task directions, and different kinds of UML diagrams, we believe that our findings have significant value to the software engineering community. The rest of the paper is structured as follows. Section 2 discusses related work in the area of consistency checking and similar experiments. Section 3 presents the experimental set-up in-depth and the variables that were taken into consideration. Section 4 reports the results and the findings of the experiment. In Section 5, we discuss the threats to validity and how threats were mitigated or avoided. Finally, Section 6 concludes this paper and discusses possible areas of future research.

2 Related Work

To the best of our knowledge, no empirical study ever investigated the benefits of inconsistency feedback for engineers between code and models. However, there is significant related work on consistency checking mechanisms and consistency rules. Krishnan and Kellner [KK99] reported the relation between software defects and the lack of software process consistency in an empirical study. The study was based on 45 products from leading software vendors which adopted various practices of the Capability Maturity Model (CMM). Their focus was on the number of produced defects in a product when the vendors do not consistently follow the processes described in the CMM. The authors found that consistent adoption of CMM practices and personal capability of the team reduce the number of product defects in a developed system.

Mäder and Egyed [ME12, ME15] conducted a similar experiment to measure if traceability benefits software developers when they evolve and maintain a software system. [ME15] extended the work presented in [ME12] and presented the effect on performance that the subjects had with the use of traces during their maintenance and development tasks. They found that effort and quality improved if subjects had available traceability. While traceability links are not the same as consistency rules, they do both imply dependencies - albeit in different ways. Hence, it can be argued that any manner which helps developers understanding relationships say between model and code, is useful.

Spanoudakis and Zisman [SZ01] discovered that different stakeholders produced inconsistencies in overlapping models in their survey. They presented their findings as parts of a process for identifying and resolving inconsistencies. They also presented processes for identifying overlapping models and, how to diagnose inconsistencies on these overlapping models. Finally, they described the handling, tracking and management of inconsistencies for these overlapping models.

Usman et al. [UNhKsC08] presented consistency checking techniques for UML models in their survey. They found that nearly all consistency checking techniques are based on rules and monitor these rules on the UML models. They analyzed each tool with their defined parameters e.g. Nature of consistency checking, UML diagrams, UML version etc. and they classified the techniques according to their intermediate representation.

These studies have shown the potential of consistency checking and fixing mechanisms but they did not show the effect for engineers when correcting inconsistencies between model and code. This is the objective of our empirical study to explore any benefit of such mechanisms. Thus, the nature of our study differs from the previous work described in this section.

3 Method

This section details the method used in our controlled experiment with student subjects and allows its reproducibility. The subjects had to manually complete change requests between model and code based on consistency rules provided to them. For some of these change requests, we provided inconsistency feedback. For others we did not. In total, subjects had to complete ten such change requests with different degrees of complexity. The task complexity was based on the number of diagram types a change request affected - simple tasks affected one type of diagram, complex tasks affected two or more types of diagrams. The premise was always the same: the subjects were given an already changed model or code together with a change description; and they were asked to propagate the changes to the unchanged code or model. This is where the consistency information came into play. Inconsistency feedback becomes relevant after the model or code changes. The experiment thus explored if the availability of inconsistency feedback made a difference in propagating the initial changes.

We define inconsistencies as situations where engineering artifacts are contradictory. Since, this experiment focused on model and code, inconsistencies imply that model and code are out of synchronization. Our working assumption was that the initial change to the model or code was correct and the inconsistencies was the result of not yet having propagated the change to the code or model respectively.

Inconsistencies are defined through consistency rules - which describe a condition that must hold [UNhKsC08]. Since our experiment focused on models and code (specifically UML class, state and sequence diagrams as well as Java source code), we considered consistency rules between model and code applied in this experiment, Table 1 shows these six rules. For example, for a sequence diagram the order of calls

Table 1 – Consistency rules used in the experiment from Reidl et all. [RDE14]

Diagram Type	**Consistency rules**
Class Diagram Consistency Rules	CR1: The structural features of the class diagram to be represented in the code (vice versa)
	CR2: The code class references and inheritance must be present in the class diagram (vice versa)
Sequence Diagram Consistency Rules	CR3: Order of calls in the Sequence Diagram must correspond to the code (vice versa)
	CR4: Methods' calls from a code class must be represented to the lifeline of that class (vice versa)
State chart Diagram Consistency Rules	CR5: The transition between states are methods with condition and must be represented in the code (vice versa)
	CR6: The code execution state space must follow the state chart diagram's state sequence (vice versa)

among messages must correspond to the sequence of calls in the code.

A change request came in two variations: either the code was changed by us and the subjects were asked to resolve inconsistencies with the model; or the model was changed by us and hence subjects had to resolve inconsistencies with the code. Regardless of task, project or change direction, the consistency rules were always the same.

To control the experiment, a laboratory was used. The allocated time slot for the experiment was three hours. Since the laboratory could not accommodate all subjects at once, we performed the experiment three times in a row with roughly 12 subjects each. The data were analyzed as one set because each setup was identical and this experiment should not be considered a family of experiments [BSL99] but rather as one experiment.

The three groups' experiment parameters were kept constant. A procedure was set up that followed a predefined schedule. Instructions were given to subjects orally. A demo change request was done together with the subjects to familiarize them to the tasks. An initial session of 20 minutes was reserved for this. Subjects could ask questions. Since, we considered each subject's individual effort and correctness as part of the experimental analysis, subjects were not allowed to interact with one another during the experiment.

3.1 Subjects

The subjects performed the experiment as a supplementary part of a software engineering course. We asked them to volunteer for this experiment and their participation was not a mandatory part of their course. We did not exclude any volunteering student from the experiment. We had 36 subjects, all bachelor-level computer science students at our university: 31 were male and five were female. Their mean age was 23.7 years with the youngest subject having been 20 years old and the oldest subject having been 38 years old. Their professional programming experience ranged from 0 years (no experience) to 17 years. The average professional experience for the subjects was two years. As computer science students, all had extensive academic programming experiences; all were familiar with Java and UML. The subjects had an average Java experience of three years and about one year of UML modeling experience. The results

Table 2 – Summary of Projects in Model and Code Size

Project	Lines of Code	Number of Classes	Different Diagrams	Diagrams	Model Elements
Matador	5.674	37	Class, Sequence, Statechart	6	661
Calendarium	21.016	150	Class, Sequence, Statechart	12	2.843

from these subjects can be generalized for the target group of junior engineers.

3.2 Independent Variables

The experiment aimed to measure the effort and correctness of subjects while they resolved the inconsistencies stemming from the change requests. Every subject received the same change request tasks - half the tasks with inconsistency feedback and the other half without inconsistency feedback. Every subject thus had tasks with and without feedback. Furthermore, every subject had to work on both change directions - half the tasks focusing on model-to-code changes and the other half focusing on code-to-model changes. This setup avoided accidental groups of "good" subjects or "bad" subjects.

We prepared ten tasks for two different projects to ensure that our findings were not limited to a specific project or a specific task. Each project had five tasks. Each task involved types of UML diagrams. Moreover, the tasks were of different complexity. Two tasks affected one UML diagram type; two tasks affected two UML diagram types and one task affected all three UML diagram types. Each change request task has a model-to-code and code-to-model direction with or without inconsistency feedback.

Hence, the experiment's independent variables were: *the project*, *the change direction*, *the (availability of) inconsistency feedback* and *the complexity of the tasks*. The independent variables are described in detail next.

Projects We used two different projects to ensure that the results are applicable beyond a specific project. The first project was an open source monopoly game called "Matador" and the second project was a proprietary calendar application called "Calendarium". Both projects were implemented in Java and they were selected because of the availability of source code and different UML diagrams types.

The game project *Matador* is a Monopoly-like game with some alterations from the original Monopoly game. It is quite small with <6KLOC across 37 Java classes. Altogether with six diagrams and 661 model elements were available. The calendar application *Calendarium* is larger with 21KLOC across 150 Java classes. It has 12 UML diagrams and 2843 model elements. Nevertheless, the size of these project was ideal for a three hours experiment while being diverse and complex enough. Note that the sizes of the projects, as seen in Table 2, vary among tasks because of the changes we made to trigger the experiments. The data set is archived in the FigShare platform[1] to be used for reproducibility purposes.

Change Direction In model driven engineering, programmers work side-by-side with modelers. A modeler could thus initiate a task by changing the model which the

[1] https://figshare.com/s/cb1a69751b68de5cc6a7

Table 3 – Example of inconsistency feedback per rule provided to the subjects

Inconsistency	**Example message**
Violation of CR1	Methods getTransaction() and getConnections are missing in the class ConnectionManager in "dblayer class diagram"
Violation of CR2	Missing relationship to Interface Controller from classes Game Controller, Board Controller, Player Controller ...
Violation of CR3	Messages lock(), commit() and processSql() in "DBLayer sequence diagram" are called in the wrong order as it is the code in the method deleteEntityObject()
Violation of CR4	Messages lock(), commit() and processSql() in "DBLayer sequence diagram" are called from the wrong lifeline
Violation of CR5	Transitions in the "PersistentBroker State Machine diagram" are not declared as methods in the class PersistenceBroker
Violation of CR6	State generated by the payDept() not present in the statechart diagram "Matador_Statecdiagram"

programmer then needs to implement in the code. Or a programmer could perform the requested changes first which the modeler will then document in the model. Consequently, inconsistencies may arise from either model changes or code changes in both directions. Regardless, these inconsistencies need to be resolved. Thus, for each of the ten tasks, we created two change directions: a task could either be initiated by model changes that need to be propagated to code; or by code changes that need to be propagated to the model. Each subject was randomly assigned a change direction for each task. Every subject thus had five model to code and five code to model change requests.

Inconsistency Feedback The experiment examined the impact of having knowledge about inconsistencies between model and code. Thus, another independent variable was the availability of such knowledge. We considered consistency rules that cut between model and code. Consistency rules imposed by the specific languages for their artifacts are not violated by the created tasks and they are respected by the participants. Table 1 presented the rules that must hold between the model and code. Table 3 presents example messages for each consistency rule that subjects had to consider as part of the inconsistency feedback.

Complexity of Task We choose ten change request tasks with increasing complexity. We define the task complexity to be related to the number of diagrams a task is affecting. Multi diagram tasks are more complex than one diagram tasks since multiple elements must be consistent not only between model and code but also across the affected diagrams. On the code side, multi diagram tasks usually affect multiple classes and methods in them in order to achieve consistency between the model and the code. Four change requests were designed to affect only one kind of diagram, e.g., class diagram only or statechart diagram only. Four more change requests were designed to affect two types of diagrams, e.g., class and sequence diagrams or sequence and statechart diagrams. The last two change request tasks were designed to affect all types of diagrams, e.g., class, sequence and statechart diagrams. The tasks covered actual change requests that can occur in a project development phase where change can affect multiple type of model diagrams and code places. Every task is designed to be totally independent from the others task of the project.

3.3 Dependent Variables

The experiment measured the performance of subjects resolving inconsistencies when inconsistency feedback was provided in contrast to when it was not provided. To measure the performance, we observe two variables: 1) *the time (effort)* subjects required to perform each change request task and 2) *the correctness* of each provided solution. The time variable describes the duration a subject required to complete a task. The time was measured in minutes and it was calculated by subjects recording the start and end times for every task they worked on.

The correctness variable describes the correctness of the given solution with respect to resolved existing inconsistencies. Correctness is a categorical variable with three discrete values, *incorrect, partially correct and correct.* Each solution required multiple changes to be performed for the solution to be complete. We split the changes into three groups of related changes (e.g, method renaming was one group, class relationships was another group, etc.). Subsequently, we assessed how many of these groups the subjects had completed. Tasks for which subjects did not complete any group were considered incorrect. Tasks for which one or two groups were completed were considered partially correct. Finally, tasks for which all three groups of changes were completed were considered as correct. Correctness was assessed by three people, two authors and an external person.

3.4 Experiment Variations

The experiment was designed with two projects in mind that every subject would have worked on, each with five change request tasks. Each task has a direction (model change to code or code change to model). A change direction was either supplemented with inconsistency feedback or it was not. Hence, there were 2x2x5 = 20 possible settings how the experiment could unfold (following the factorial design [BHH05]). While the tasks were randomly assigned to subjects, we ensured that all possible task variations were included. However, every setting had the tasks of each project being grouped together. Thus, a subject could start with Matador or Calendarium project and had to complete its tasks before the subject could move to the next project.

3.5 Experiment Tool

The experiment was conducted using the Eclipse Integrated Development Environment (IDE)[2] in the Java development edition. We installed the UML Designer plug-in[3] for the UML model of the projects. We did not use any other tools for the experiment since the Eclipsed IDE with the modeling plug-in could serve both developer roles (Modeler and Programmer). The use of a standard IDE was preferred over the development of a specialized tool due to the support for both types of work artifacts for the experiment and the familiarity this tool had among all subjects.

[2]Eclipse IDE website: https://eclipse.org/
[3]Plug-ins website https://marketplace.eclipse.org/content/uml-designer

3.6 Procedure and Material

Before the start of the experiment, we provided informational material to subjects. The material contained a questionnaire about the demographic information of the subjects. We also provided a demo task to illustrate an example of the problem and the solution (another task which was not used in the experiment but served for training purposes only). Finally, the subjects received the list of ten tasks they had to complete. The experiment did not require the subjects to be aware of the functionality of the projects. We spent the initial 20 minutes of the experiment for a briefing. The briefing included the subjects filling in the required demographic information and then proceeding to explaining the purpose and scope of the experiment. This briefing also included, a presentation about the Eclipse IDE and its modeling perspective for the UML model of the task. During the presentation of Eclipse IDE, we also discussed the demo task to provide a concrete example.

Main Experiment Part Following the briefing, the subjects started the main phase of the experiment. In this phase, each subject had to solve the ten tasks in the assigned order. Initial test runs (conducted with other students beforehand) showed that the tasks were solvable in roughly three hours. Therefore, we considered three hours working time to be sufficient for the experiment. For each task, subjects had to record the start and end times on their own task sheets. The changes had to be implemented using the Eclipse IDE. Both the task sheets and the workspaces of the Eclipse IDEs (this is where the model and code is stored) were collected at the end of the experiment to measure time and correctness.

After the experiment The final step was to check the answers provided by the subjects for correctness and to record the data in an excel sheet for further statistical analysis. In this step, we filled the column with the correctness value after checking one by one the answers of the subjects. Correctness was checked by three people independently. Differences were discussed and resolved to derive a single answer for each task: Correct, partially correct, or incorrect. The recorded time was also stored in the same excel sheet for the analysis phase. Unfinished tasks were rejected from the analysis process since they could not be measured in respect to time and correctness.

4 Analysis and Results

This section analyses the results we gathered from the performed experiment. The research questions provide the motivation for the following analysis of the results.

4.1 Research Questions

In this experiment we are investigating the effect of inconsistency feedback provided to engineers resolving inconsistencies. We will answer the following research questions.

1. What is the effect of inconsistency feedback on correctness when engineers are presented with inconsistent models and code changes?
2. What is the effect of inconsistency feedback on time (effort)?
3. Does the complexity of the tasks affect time and correctness?

Table 4 – ANOVA analysis of the effort depended variable Time

Variables	**Df**	**Sum Sq**	**Mean Sq**	**F value**	**Pr (>F)**	**Signif.**
Feedback	1	161.6	161.62	4.8151	0.029199	*
Direction	1	1	0.8	0.024	0.87679	
Project	1	174.1	174.09	5.1864	0.023671	*
Complexity	9	169	18.7	0.581	81181	
Feedback:Direction	1	329.6	329.61	9.8198	0.001948	**
Feedback:Project	1	293	293.1	9.093	0.0029	**
Residuals	195	6127.3	31.422			

4.2 Overall Analysis

For the analysis of the collected data, we used the R statistical analysis environment [R D08]. We choose to analyze the dependent variable time with the ANOVA [DCF$^+$16] method for the analysis of variance. ANOVA is based on the assumptions of normal distribution and homogeneity of variance. To test this, we performed the Shapiro-Wilk's [SW65] test for data normality in the continuous depended variable time (W=0.99408, p-value=0.4749, $\alpha = 0.05$). The test provides the null hypothesis that the data are normal and the p-value confirmed it.

Since, the data was normally distributed, we performed the Bartlett's homogeneity test [SC89](K-squared = 5.3777, p-value = 0.1461) which provided no evidence for lack of the variance homogeneity. We have considered the following formula for the ANOVA analysis for the time depended variable

$$Time \sim Feedback * Change\ Direction * Project + Task\ Complexity \quad (1)$$

For the ANOVA analysis, we observe six different columns, the degree of freedom (Df) which is the number of freedom of a variable, the sum of squares (Sum Sq) is the sum of squares of the deviations of values from the expected value, the mean squares (Mean Sq) is the division of Sum Sq by the Df, the F value which is the ration between two mean squares that forms the basis for a hypothesis test, the p-value the probability to obtain the statistic F value if the null hypothesis is true and the Significance(Signif) where it shows the effect of a independent variable to the outcome of a dependent variable.

In Table 4, we see that the inconsistency feedback independent variable has an effect on the time dependent variable in correlation with project and direction independent variables. This means that the time is affected by the provided feedback both when the project changes and when the direction is different. We need to examine in the detailed analysis to find out in what way time is affected. In this ANOVA analysis, we also see a variation in time between the subjects resolving tasks in the different projects. We expected that due to the difference in size and nature of the two projects. We also performed ANOVA on the second depended variable correctness. The formula used for the ANOVA is

$$Correctness \sim Feedback * Change\ Direction * Project * Task\ Complexity \quad (2)$$

In Table 5, we observe that the variance of correctness is mostly affected by the inconsistency feedback independent variable. This is interesting since multiple other variables are changing but they do not seem to have a large effect in the correctness

Table 5 – ANOVA analysis of the Correctness dependent variable

	Df	Sum Sq	Mean Sq	F value	Pr (>F)	Signif.
Feedback	1	13.69	13.695	25.137	1.19E-06	***
Direction	1	3.59	3.587	6.584	0.011	*
Project	1	0.89	0.892	1.638	0.2021	
Complexity	9	11.97	1.33	2.441	0.0118	*
Feedback:Direction	1	0.01	0.01	0.019	0.8912	
Feedback:Project	1	0	0.003	0.006	0.938	
Direction:Project	1	0.25	0.246	0.452	0.5024	
Feedback:Complexity	8	4.23	0.528	0.97	0.461	
Direction:Complexity	8	8.72	1.089	2	0.0483	*
Feedback:Direction:Project	1	0.64	641	1.176	0.2795	
Feedback:Direction:Complexity	8	1.39	0.174	1.176	0.9579	
Residuals	197	107.33	0.545			

Table 6 – Overall Correctness

Correctness	Inconsistency Feedback	No Inconsistency Feedback	Incr
Correct	59(41.84%)	22(15.71%)	268%
Partially Correct	49(34.75%)	50(35.71%)	0%
Incorrect	33(23.4%)	68(48.57%)	-51,5%

depended variable. The only ones that have some effect into the dependent variables are the direction and complexity. Complexity was expected to have some effect since our subjects have different level of experience, providing an interest point towards the effect of direction to correctness.

After the analysis of the variance for the two dependent variables, we present the overall time and correctness results of the subjects participated in the experiment. In total, we had 360 tasks: 180 with inconsistency feedback and 180 without inconsistency feedback. The subjects completed 141 tasks with inconsistency feedback and 140 without feedback. In order to answer our first research question, we present the analysis in the correctness variable in subsection 4.3 where we examine overall correctness and correctness per independent variable.

4.3 Research Question 1

Revisiting our first question, we are interested in identifying the effect of inconsistency feedback for subjects correctly resolving change tasks. Table 6 presents the correctness of the subjects in fixing inconsistencies at the requested tasks. We observe that subjects with inconsistency feedback have solved almost three time more correct tasks than without inconsistency feedback. This overall 268% increase is seen by having 59 correct tasks for subjects with inconsistency feedback and only 22 correct tasks for subjects without inconsistency feedback. The partially correct tasks remained at the same levels for subjects with and without inconsistency feedback.

Table 6 showed overall correctness. In the following, we will further examine the correctness with respect to the different projects and the change request direction. We expect from the variance analysis above that we would not have any significant difference from the overall analysis for the independent variables Projects and Direction

Table 7 – Correctness per Project

Matador	**Inconsistency Feedback**	**No Inconsistency Feedback**	**Incr**
Correct	34(44.74%)	15(20%)	226,7%
Partially Correct	18(23.68%)	24(32%)	-25%
Incorrect	24(31.58%)	36(48%)	-33,4%
Callendarium	**Inconsistency Feedback**	**No Inconsistency Feedback**	**Incr**
Correct	24(36.92%)	7(10.77%)	342%
Partially Correct	26(40%)	26(40%)	0%
Incorrect	15(23.08%)	32(49.23%)	-46,9%

Table 8 – Correctness per Direction

Model to Code	**Inconsistency Feedback**	**No Inconsistency Feedback**	**Incr**
Correct	40(47.06%)	14(23.33%)	285,7%
Partially Correct	30(35.29%)	23(38.33%)	130%
Incorrect	15(17.65%)	23(38.33%)	-65%
Code to Model	**Inconsistency Feedback**	**No Inconsistency Feedback**	**Incr**
Correct	19(33.93%)	8(10.00%)	237,5%
Partially Correct	19(33.93%)	27(33.75%)	-70,4%
Incorrect	18(32.14%)	45(56.25%)	-40%

seen in Table 5.

Table 7 verifies the findings from the ANOVA analysis about the correctness improvement due to inconsistency feedback. We see that in both projects, inconsistency feedback improved task correctness more than 200% with partially correct task remaining in the same levels for both groups. It is interesting to observe that the correctness with inconsistency feedback is better for the larger project "Calendarium" than compared to the smaller project "Matador". This suggests that the importance of inconsistency feedback improves with project size but more data is needed to confirm this.

Finally, as can be seen in Table 8, the different change directions do not affect the correctness of inconsistency feedback regardless of direction. The benefit is respectively 285% and 237% for both model to code inconsistencies and code to model inconsistencies. Thus, we conclude that both designers and programmers can benefit equally from inconsistency feedback in keeping model and code consistent.

4.4 Research Question 2

Next, we are discussing the effect of inconsistency feedback on the effort that engineers need in identifying and resolving inconsistencies. We measure the effort with the time subjects needed for completing their tasks.

In Table 9, we see that the mean overall time that subjects required to complete the tasks without inconsistency feedback is in fact smaller than those with inconsistency feedback. This is the case for all above scenarios for tasks (all task, correct tasks, correct and partially correct tasks). In the case of the correct task, we also observe

Table 9 – Overall Time Measurements

Mean Time (min)	Inconsistency Feedback	No Inconsistency Feedback
All Tasks	21.86	19.11
Correct Tasks	22.90	22.64
Correct & Partial Correct	22.94	20.46

Table 10 – Mean Time per Project for Correct and Partial Correct

Mean Time(min)	Inconsistency Feedback	No Inconsistency Feedback	Diff
Matador	24.34	20.87	3.47
Callendarium	21.9	19.97	1.93

that the medium time is almost identical. We will explore this phenomenon with the time variable further down in subsection 4.5 where we present a more detailed analysis on the effort required by the subjects to complete the tasks with and without inconsistency feedback.

Table 10 presents the mean time of the subjects completing task correct or partially correct between the two projects. We observe that in both projects the mean time of the subjects is slightly higher for those having inconsistency feedback than those without inconsistency feedback. We expected a variation between the subjects with and without inconsistency feedback based on the variance analysis of the time dependent variable when correlated to the inconsistency feedback.

According to the ANOVA analysis, we expect a variation for the time measurements between the different directions of the tasks. In Table 11, we observe that the model to code change request tasks have benefited in time from the inconsistency feedback which was expected since a model change usually is required to be implemented with a specific way to the code. Interestingly, the code to model direction is far from being considered benefited by the inconsistency feedback since there is large time difference between subjects with inconsistency feedback and subjects without it.

In order to understand what is happening with the time measurements, we are going to limit the comparison groups to only correct tasks to see if the same results are present.

We again observe in Table 12 that subjects without feedback took less time than the subjects with inconsistency feedback. This is interesting because we expected before the experiment that the subjects with the inconsistency feedback will require less effort to complete the tasks. Thus, we need to examine deeper into the correct tasks to reveal the causes of such diversion from our expectation. From Table 10 and Table 11, we see that the time was not favorably affected by inconsistency feedback. However, not all tasks have the same complexity. Therefore, we investigated more carefully how time and correctness are influenced by the tasks' complexity. The findings are presented into the subsection below where we present the effect of complexity to the effort of subjects completing tasks and what kind of tasks were able to complete with and without inconsistency feedback.

Table 11 – Mean Time per Direction for Correct and Partial Correct

Mean Time(min)	**Inconsistency Feedback**	**No Inconsistency Feedback**	**Diff**
Model to Code	19.88	24.43	4.55
Code to Model	28.5	16.26	12.24

Table 12 – Mean Time per Project and Direction for only Correct Tasks

Mean Time per Project (min)	**Inconsistency Feedback**	**No Inconsistency Feedback**	**Diff**
Matador	22.06	23.8	1.74
Callendarium	24.3	20.14	4.16
Mean Time per Direction (min)			
Model to Code	20.08	14.21	5.87
Code to Model	28.68	17.75	10.93

4.5 Research Question 3

Initially, we examined the effort subjects used based on the complexity of the tasks in order to investigate their effect on the mean time for the subjects with inconsistency feedback. In our experiment, we included tasks that are of different complexity. For each project, we have two tasks that are simple and affect only one kind of diagrams, two that affect two kinds of diagrams and one task that affects all three kinds of diagrams in the model. We grouped these tasks into simple tasks (i.e. one diagram type affected) and complex tasks (i.e. two or three diagram types affected).

We also observed in Table 9 that the time subjects took to complete tasks are relative better for those without feedback. However, this does not unveil the effect of the tasks' complexity on the overall mean time for correct tasks.

In Table 13, we present the correct tasks divided by their different complexity with and without inconsistency feedback. We see that the number of complex tasks being resolved correctly are higher (31) with consistency feedback than without inconsistency feedback (10). We also observe that when feedback was provided, the number of complex correct tasks is higher than the simple ones (31 vs 28). Whereas without inconsistency feedback the number of complex tasks is lower than the simple tasks (10 vs 12). Table 14 similarly shows that the mean time for tasks divided by their different complexity with and without inconsistency feedback. It shows that for simple tasks when inconsistency feedback is not provided, their mean time is significantly lower than when inconsistency feedback is provided. This explains the overall mean time being relatively better for the subjects without inconsistency feedback since the simple tasks were the most resolved tasks in that case. Herein, the overall mean time without inconsistency feedback is mainly affected by the time subjects required for simple tasks.

Furthermore, in Table 14 we see that the mean time for correct complex tasks is lower for subjects with inconsistency feedback. This suggests that subjects benefited by inconsistency feedback when resolving complex tasks. However, when inconsistency feedback was not provided, the number of correct complex tasks per affected types of diagrams is statistically insufficient (e.g., two tasks for class*sequence) to draw conclusions.

Table 13 – Correctness for the different Complexities of Tasks

Correct Task Analysis	**Inconsistency Feedback**	**No Inconsistency Feedback**
Class	9	3
Sequence	12	6
State	7	3
Total	**28**	**12**
Class*Sequence	10	2
Class*State	3	2
Sequence*State	5	2
Class*Sequence*State	13	4
Total	**31**	**10**

Table 14 – Mean Time for the different Complexities of Tasks

Correct Tasks Mean Time	**Inconsistency Feedback**	**No Inconsistency Feedback**	**Diff**
Class	23.1	18	-5.1
Sequence	17.5	12.6	-4.9
State	25.1	28.33	3.23
Total Mean Time	23.71	13.17	
Class*Sequence	22.3	68	45.7
Class*State	23	23.5	0.5
Sequence*State	23.2	15.5	-7.7
Class*Sequence*State	21.3	20	-1.3
Total Mean Time	22.1	28.30	

Figure 1 presents the number of correct complex tasks based on the different UML diagram types when providing inconsistency feedback or not. There are four types of tasks that affect multiple diagrams. The tasks for all three diagrams and for a class and sequence diagrams are present in both projects. The class*statechart task is present only in the “Callendarium” project while the sequence*statechart is present only in the “Matador” project. This explains their lower number in Figure 1 in contrast to the other two combinations. We choose to have the class*sequence task interleaving between the two projects because there are the most commonly used diagrams in UML modeling. The class*sequence*statechart task is also present in both projects since it is the most complex task. Figure 1 shows that all different complex tasks benefit from inconsistency feedback. In particular, the correct class*sequence tasks and class*sequence*statechart were respectively five and three times higher when inconsistency feedback was provided.

We performed the same analysis for the simple task to see which UML diagram type is more affected by the inconsistency feedback. We have class diagram tasks present only in the “Callendarium” project while we have statechart diagram tasks present only in the “Matador” project. Sequence diagram tasks are present in both projects. Figure 2 shows the number of corrects simple tasks per diagram type. We have twice as many sequence diagram tasks compared to the other two diagram types, because we included this task type in both projects. Another interesting observation is that sequence and statechart diagrams are favored around two times when inconsistency feedback is applied while the class diagram type tasks are improved by three times. This subsequently leads to class diagrams being the most favored among simple tasks

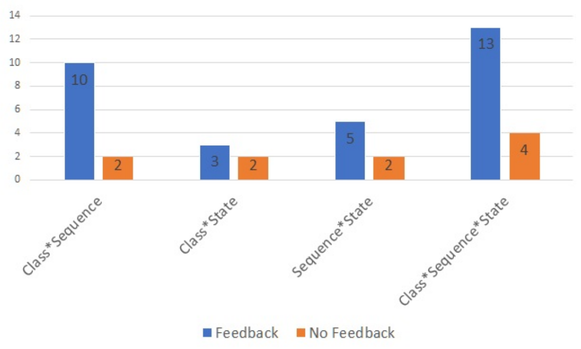

Figure 1 – Correct Complex Task Analysis

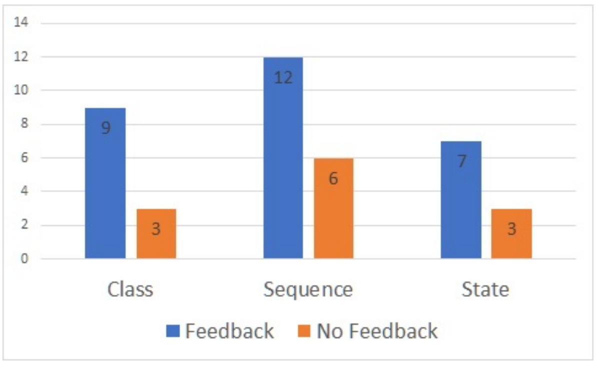

Figure 2 – Correct Simple Task Analysis

when inconsistency feedback is present. Figure 2 shows that all different simple tasks benefit as well from inconsistency feedback.

5 Threats to Validity

In this section, we discuss the threats to validity ([FM10], [HP00]) of this study and explain how we mitigated these dangers in this experiment.

5.1 External Validity

A possible external validity threat is the use of students as subjects in the experiment. Recent studies [SMJ15, SAW08] have shown that students can be valid subjects for experiments and that students are well representative when it comes to developing tasks. To reduce the effect of this threat, we included students with a variety of professional and academic experience in programming and modeling. We had subjects with minimum knowledge of programming and modeling and subjects which are considered experienced programmers.

From the subjects used in the experiment, one may conclude that our generalization group is junior engineers. This limitation is present in the use of students as subjects but we expect that consistency feedback will be beneficial to senior engineers working on larger projects. Thus, we expect to have similar findings for also more experienced engineers but reproducing the experiment with practitioners is required to make conclusions.

Another possible threat is the chosen projects. We chose to have two projects which were both implemented in Java but we expect that this does not affect the results in any other programming languages since consistency checking does not focuses on programming language specific aspects but rather on the existence of key modeling aspects that should also be found in the code.

Our study was performed on projects that were medium and large for the scope and the three hours time subjects were asked to participate. Both projects had models that consisted of three type of diagrams: class, sequence and statechart diagrams. The projects were applications in different nature, a game and a calendar, thus, we can avoid the possibility of the results to be applicable only to the specific nature of a project. The choice to avoid larger more complex ($>$ 1M LOC) projects was deliberately taken because in complex, real-world systems, engineers often have in-depth knowledge of the system they are designing and maintaining. Thus, they are likely to know where to propagate changes when inconsistencies arise. By choosing smaller projects, we aimed at mitigating the subjects' unfamiliarity and difficulty to understand the projects.

However, the fact that we found strong benefit in correctly resolving inconsistencies in projects of this size, implies that more complex projects will be equally (or better) affected by the use of consistency feedback to resolve inconsistencies. We also believe that in more complex projects, we can gain effort as well since the complexity of the inconsistencies will be higher than in the projects used in our experiment.

Finally, we choose to use a real life IDE for the experiment in order to imitate real working environments and condition for the experiment and the subjects.

5.2 Internal Validity

To avoid variability in knowledge between the subjects, we made sure not to provide any information about the experiment prior to the date of the experiment. All material was made available to the subjects during the beginning of the experiment and they were kept at a minimum. We asked the subjects after the briefings if they had any questions but apart from some organizational questions, they claimed to be ready for the experiment. The subjects had no prior knowledge of the projects. They could also not know about the tasks which were created by us for the purpose of this experiment.

To avoid the Hawthorne effect, we used subjects from a course that the authors were not tutor or lecturers so that the subjects will not feel examined from their teacher. We also during the briefing and invitation to participation that their performance in the experiment will not be assessed within their course. We confirm them that their performance was only relevant for metrics within our research group and for only research purposes.

The design of the study is crossover study, a variant of within-subject design in which all participants are exposed sequentially to all variances of the tasks. This design minimizes the number of participants required to identify statistically significant differences between the results [JK14]. This design has the learning effect as a threat. This threat was mitigated by assigning the tasks to the subject in different order. Thus, one subject could get a task as first whereas another subject would get it as last. All tasks were assigned in all different places.

The time that subjects could use to solve the tasks was limited to three hours in total without limitation per task. The time limit was chosen by preliminary test runs (with different subjects). Most of the 36 subjects have completed all of their task, which suggests that the assigned time was sufficient for the average subject. We also did not allow the students to communicate with each other and we provided different set up for each one of them.

We had created the initial changes and inconsistencies, allowing us to control the complexity of each task based on the affected UML diagram types. To reduce the threat of meaningless change requests we created real life inconsistencies from Table 1 between code and models. The goal was to investigate the effect of consistency feedback on the resolved tasks.

Finally, to avoid that completed tasks were incorrectly categorized, we performed the correctness check independently by three different people. In case of diverse opinion among the three, a discussion session was held for concluding the correctness of the specific task.

5.3 Construct Validity

The experiment aimed to evaluate the effect of consistency feedback in resolving inconsistencies between collaborating software modelers and software developers. The problem is caused when the selected dependent variables do not relate to the target of the experiment. Inconsistencies usually require the developer's time to be resolved and not all of them are surely resolved correctly. Thus, we chose to assess the effect by measuring the effort and correctness of the subjects resolving the introduced inconsistencies. We measured the effort with the time subjects required to complete each task. Faster time in resolving inconsistencies implies less effort by the subject whereas correctness expects the subjects to resolve all inconsistencies between model and code based on the defined consistency rules.

5.4 Conclusion Validity

Possible threats to the statistical conclusion validity are low statistical power, low effect size and violation of the assumptions for the statistical procedure. To avoid these threats, we worked with 36 subjects, created equally sized groups with randomized task assignment. Before the data analysis, we verified the normality assumption using the Shapiro-Wilk test and we used the Bartlett's homogeneity test to verify homogeneity over the time data. Later, we performed the ANOVA analysis of time and correctness variables. However, for a finer grain analysis, such as for the effort (time) of subjects for complex tasks, we need to reproduce the experiment and gather more data points. This is future work.

6 Conclusion

In this paper, we investigated the usefulness of inconsistency feedback during the co-change of models and code. We conducted a controlled experiment with 36 subjects that were asked to resolve inconsistencies between model and code. We designed ten change requests tasks of different complexity for two diverse in nature projects. Each task has two directions of co-change one from model to code and one from code to model. Every direction has two variations with consistency feedback provided or not.

For each project, we had a model with three types of UML diagrams (class, sequence, statechart diagrams) and its Java source code.

We found that consistency feedback provides a significant effect in correctly resolving inconsistencies. Subjects with consistency feedback resolved correctly 268% more tasks than those without consistency feedback (59 with consistency feedback versus 22 without). This effect was similarly observed for any direction and any project. Effort was not significantly affected in the overall analysis, leading us to explore deeper the causes for it. Thus, we examined the cause of the time not being affected by consistency feedback focusing on the complexity of the tasks. We found that subject with consistency feedback have resolved correctly three time more complex tasks (31 over 10). Interestingly, with consistency feedback subjects resolved more complex tasks than simple tasks (31 vs 28). Whereas without consistency feedback subjects resolved more simple tasks than complex tasks (12 vs 10). The mean time for simple tasks was significantly lower when consistency feedback was not provided. This resulted in the overall mean time being relatively lower for subjects without consistency feedback. However, the mean time for complex tasks was lower when consistency feedback was provided.

Our findings provide evidence that the consistency feedback, when present, can benefit developers and designers in correctly resolving inconsistencies. In particular when correcting more complex tasks. As future work, we plan to reproduce our experiment to gain more evidence especially to further investigate the effect of consistency feedback on time. At the second experiment, we will add an additional post-study questionnaire to see the level of trust from subjects to the consistency feedback provided.

References

[BHH05] G. E. P. Box, J. S. Hunter, and W. G. Hunter. *Statistics for Experimenters*. Wiley, 2005.

[BSL99] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473, Jul 1999. doi:10.1109/32.799939.

[DCF+16] Peter K Dunn, Michael D Carey, Michael B Farrar, Alice M Richardson, and Christine McDonald. Introductory statistics textbooks and the gaise recommendations. *The American Statistician*, (just-accepted), 2016.

[Egy11] Alexander Egyed. Automatically detecting and tracking inconsistencies in software design models. *IEEE Transactions on Software Engineering*, 37(2):188–204, 2011.

[FM10] Robert Feldt and Ana Magazinius. Validity threats in empirical software engineering research - an initial survey. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*, pages 374–379, 2010.

[HP00] Marilyn Healy and Chad Perry. Comprehensive criteria to judge validity and reliability of qualitative research within the realism paradigm. *Qualitative Market Research: An International Journal*, 3(3):118–126, 2000. URL: http://dx.doi.org/10.1108/13522750010333861, doi:10.1108/13522750010333861.

[JK14] Byron Jones and Michael G Kenward. *Design and analysis of cross-over trials*. Chapman and Hall/CRC, 2014.

[KK99] M. S. Krishnan and M. I. Kellner. Measuring process consistency: implications for reducing software defects. *IEEE Transactions on Software Engineering*, 25(6):800–815, Nov 1999. doi:10.1109/32.824401.

[ME12] P. Mäder and A. Egyed. Assessing the effect of requirements traceability for software maintenance. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 171–180. ICSM 2012, Sept 2012. doi:10.1109/ICSM.2012.6405269.

[ME15] Patrick Mäder and Alexander Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, 20(2):413–441, 2015. URL: http://dx.doi.org/10.1007/s10664-014-9314-z, doi:10.1007/s10664-014-9314-z.

[MPC$^+$02] Madanlal Musuvathi, David Y. W. Park, Andy Chou, Dawson R. Engler, and David L. Dill. Cmc: A pragmatic approach to model checking real code. *SIGOPS Oper. Syst. Rev.*, 36(SI):75–88, December 2002. URL: http://doi.acm.org/10.1145/844128.844136, doi:10.1145/844128.844136.

[NEFE03] Christian Nentwich, Wolfgang Emmerich, Anthony Finkelsteiin, and Ernst Ellmer. Flexible consistency checking. *ACM Trans. Softw. Eng. Methodol.*, 12(1):28–63, January 2003. URL: http://doi.acm.org/10.1145/839268.839271, doi:10.1145/839268.839271.

[R D08] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0. URL: http://www.R-project.org.

[RDE14] Markus Riedl-Ehrenleitner, Andreas Demuth, and Alexander Egyed. Towards model-and-code consistency checking. In *IEEE 38th Annual Computer Software and Applications Conference, COMPSAC 2014, Vasteras, Sweden, July 21-25, 2014*, pages 85–90, 2014. doi:10.1109/COMPSAC.2014.91.

[SAW08] Mikael Svahnberg, Aybüke Aurum, and Claes Wohlin. Using students as subjects-an empirical evaluation. In *2nd ESEM*, pages 288–290. ACM, 2008.

[SC89] George W Snedecor and Witiiam G Cochran. Statistical methods, 8thedn. *Ames: Iowa State Univ. Press Iowa*, 1989.

[SMJ15] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. Are students representatives of professionals in software engineering experiments? In *ICSE-Volume 1*, pages 666–676. IEEE Press, 2015.

[SW65] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[SZ01] George Spanoudakis and Andrea Zisman. Inconsistency management in software engineering: Survey and open research issues. *Handbook of software engineering and knowledge engineering*, 1:329–380, 2001.

[TLG14] Damiano Torre, Yvan Labiche, and Marcela Genero. Uml consistency rules: A systematic mapping study. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 6:1–6:10, New York, NY, USA, 2014. ACM. URL: `http://doi.acm.org/10.1145/2601248.2601292`, `doi:10.1145/2601248.2601292`.

[UNhKsC08] M. Usman, A. Nadeem, T. h. Kim, and E. s. Cho. A survey of consistency checking techniques for uml models. In *2008 Advanced Software Engineering and Its Applications*, pages 57–62. ASEA 2008, Dec 2008. `doi:10.1109/ASEA.2008.40`.

About the authors

Georgios Kanakis is PhD student at the Johannes Kepler University, Institute of Software Systems Engineering. His research topic is Cloud Computer Collaboration and Change Propagation. Research interests also include software engineering, Model-Driven Engineering. Employed currently at the MIC custom solution software company in Linz, Austria. Contact him at `georgios.kanakis@mic-cust.com`

Djamel Eddine Khelladi is a CNRS researcher in the IRISA reasearch lab in the DIVERSE team, Université Rennes 1. Before that he was a Postdoctoral researcher in the Institute for Software Systems Engineering (ISSE) at the Johannes Kepler University Linz. He hold a PhD in the Laboratoire d'Infromatique de Paris 6 (LIP6) at the university of Piere et Marie Curie (UPMC). Research interests include Software engineering, Model-Driven Engineering, DSL Evolution, Evolution impacts, Software Processes, Process Adaptation, Formal Process Verification, and Agile Methods. Contact him at `djamel-eddine.khelladi@irisa.fr,djamel_eddine.khelladi@jku.at`

Stefan Fischer Dr. Stefan Fischer is a post-doctoral researcher at the Johannes Kepler University Linz. He received his Ph.D. in Computer Science from the Johannes Kepler University Linz. His main research interests are test automation and software analysis in variability intensive software systems. Contact him at `stefan.fischer@jku.at`

Michael Tröls Michael Tröls earned his master's degree in Computer Science at the Johannes Kepler University in Linz, Austria. He is currently working there as a PhD student at the Institute for Software Systems Engineering (ISSE). The focus of his work is on consistency checking within collaborative engineering environments. Contact him at `michael.troels@jku.at`

Alexander Egyed Alexander Egyed heads the Institute for Software Systems Engineering (ISSE) at the Johannes Kepler University, Austria. He received his Doctorate from the University of Southern California, USA and previously worked many years in industry before joining academia. Dr. Egyed was recognized among the Top 10 scholars in software engineering and his work has received numerous awards. Contact him at `alexander.egyed@jku.at`

Acknowledgments The research leading to these results was funded by the Land of Upper Austria - Joint International PhD Program in Informatics under the grant number Wi-2014-202477/6.
Funded by the JKU Linz Institute of Technology (LIT) the state of Upper Austria, grant no. LIT-2016-2-SEE-019 and Pro2Future, a COMET K1-Center of the Austrian Research Promotion Agency (FFG), grant no. 854184.
Received funding from Rennes Métropole under grant AIS no. 190270.