UNIVERSITY OF GOTHENBURG

A Design Science Research Study

Effects of Error Detection and Representation on the Resolution of Traceability Issues

Bachelor of Science Thesis in Software Engineering and Management

FLUTRA TAHIRAJ
MICHAEL WARNE

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

Effects of Error Detection and Representation on the Resolution of Traceability Issues
A Design Science Research Study

Flutra Tahiraj
Michael Warne

Examiner: Hang Yin

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2016

Effects of Error Detection and Representation on the Resolution of Traceability Issues:
A Design Science Research Study

Flutra Tahiraj and Michael Warne
Department of Computer Science and Engineering
University of Gothenburg,
SE-412 96
Gothenburg, Sweden
Email:flutra.tahiraj@gmail.com, michaelgwarne@gmail.com

***Abstract*—Traceability is extremely important when developing safety critical software systems. The automotive industry has strict regulations that require developers to be able to demonstrate traceability links from requirements to end products. Keeping traceability links up to date is a very expensive task; both financially and in terms of man hours. This paper conducts research into the effects of error detection and representation when traceability links become inconsistent. Additionally, we look at the effects of automating some of the process for fixing inconsistencies. The research is conducted by performing 2 experiments using a traceability management tool developed at Gothenburg University. The first investigates the use of notifications of issues and the second the use of automation when fixing issues. Our results showed that with notifications in small projects there is no significant difference when compared to finding and fixing issues manually. Our second experiment showed a significant difference in time when using automation to assist with fixing issues compared to correcting issues manually.**

***Keywords*—Traceability, requirements, software engineering, issue resolution.**

I. Purpose of Study

The IEEE [1] defines traceability as "the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match".

Traceability is considered to be a key part of software development. Not using it can lead to difficult to maintain and defective software [2]. Some industries, such as automotive, pharmaceutical and aerospace, have very strict regulations for safety critical systems that require developers to be able to demonstrate traceability of thousands of links from requirements to end products [3][4]. Keeping requirements up-to-date when systems evolve is a manual and expensive task [5]. It has been suggested, from a controlled experiment, that using traceability increases the speed of task completion and improves software maintenance quality [6].

This study has two purposes. Firstly, to research how the representation of traceability inconsistencies and problems affects their resolution. Secondly, to discover the benefits of automating fixes for the issues. To achieve this we built and evaluated, through controlled experiments, 2 extensions of a prototype traceability management tool called Capra. The extensions, known as Dash 1 and 2, enabled the detection of issues in traceability links, and the automated fixes of the said links.

Section II contains the background to the study, where we introduce Capra and Dash and explain the more technical aspects of the tool. We also cover some of the related work to this study from several systematic literature reviews. Section III explains the goal of our research, leading to our 2 research questions. Section IV describes the methodology used to create Dash, perform the experiments and analyse the data. Section V gives the results of the study while Section VI discusses them. Section VII presents our conclusions on the study and finally, Section VIII reflects on what we have learned from doing the study.

II. Background

This section gives some background information on the project we contributed to, and discusses some of the existing work that relate to our study.

A. Capra and Dash

Capra is a prototype traceability management tool being developed at Gothenburg University. It is a plug-in for the Eclipse integrated development environment (IDE) and uses the Eclipse modelling framework[1] (EMF). Capra is a tool that "is configurable, can support traceability between arbitrary artefacts and exchange of traceability information between teams and companies" [7]. In other words, the user can create and visualise traceability links between artefacts. Capra is being developed in cooperation with rt-labs[2], a small software developer based in Gothenburg and Dash is the name given to the extended versions of Capra that we created. Throughout the paper when we refer to Capra we mean the original unmodified tool, Dash 1 is the version created for both

[1]https://eclipse.org/modeling/emf/
[2]http://www.rt-labs.com/

experiments that displays warnings and Dash 2 is the version, used only in experiment 2, that implements quick fixes. For more information on the user interface see Appendix C

What follows is a description of the more technical aspects of the study.

1) Traceability Links: A traceability link has a source and target artefact; the link being the connection between the 2 artefacts. Each link appears in a trace model in the Eclipse workspace. The link itself contains references to the source and target artefacts. Capra is able to visually display the traceability links as a graph. At the time of writing this was done with PlantUml[3] but midway through the study this was being updated to a tool with more functionality. The visualisation of the trace model does not play a part in this study.

2) Artefacts and Wrappers: Capra models traceability links between artefacts in one or more projects; an artefact is any resource such as a Java class or pdf. For EMF to work with the artefacts it doesn't recognise, such as Java files, it is necessary to wrap the artefact first. For this to be possible the file type must have a corresponding handler class, e.g. Java files have a Java handler to turn them into objects EMF understands. Each wrapper is then stored in the artefact wrapper container. This container is the first of two models generated, the second is the trace model itself. The trace model contains all the links that have been created and it references the contents of the wrapper container. Each wrapper has 3 properties. A name, a handler type and a Uniform Resource Identifier (URI). In Capra a URI is similar to the file path of the resource within the workspace.

3) Issues: There are 3 specific types of issue that could occur with a traceability link and they all involve a change in the project that involves the source or targets of a link. The first is if an artefact in the project has been renamed, the second is if it has been moved and the last is if it has been deleted. All 3 make the traceability link inconsistent; the trace model will no longer match the project.

B. Related Work

There have been several researchers that have conducted systematic reviews on the subject of traceability. Nair, de la Vara and Sen [8] analyse and summarise 70 papers from 20 years of literature on the subject of traceability. They consider traceability visualisation, empirical evaluation and advanced tool support as areas for further research. Their review of traceability tools, presented in 35 papers, gives the top 3 features found as:

- Traces life-cycle: features for creating, maintaining, and updating traces between various artefacts.
- Maintaining traceability between artefacts specific to requirements specification: features for maintaining traces between requirements and managing them.
- Automated traceability: features for creating and maintaining traceability information (semi) automatically.

[3]http://plantuml.com/

Santiago et al. [9] analyse the state of the art of traceability management in the context of Model-Driven Development (MDD) and review 157 papers. While their review does cover traceability management and automation it does not include how the issues are represented in order to improve issue resolution; instead focusing on automated traceability link generation.

Torkar et al. [10] perform a review of papers from 1997 to 2007. They specifically study definitions, challenges, tools and techniques of requirements traceability. They highlight the cost of manual construction and maintenance and the perception that traceability is not financially feasible. They also discuss the issues of link maintenance from the point of view of uncoordinated teams; including the belief that traceability costs more than it delivers. While the paper discusses tools they are primarily requirements engineering tools with some traceability functionality, rather than tools designed for traceability support.

The summarised reviews contain a wealth of information on the subject of traceability and the need for tools to improve the creation and maintenance of traceability links; automated or not. There is also discussion on the visualisation of traceability. However, there is a gap in the knowledge on the benefits of tools that focus on keeping traceability links consistent; either through automation or notifications for the user.

III. Research Goal

The goal of the research is twofold. First, to investigate, via a traceability management tool, how issues in traceability links can be represented in an IDE, and how this affects both the users and usability of the tool. Secondly, we shall investigate how using automated fixes affects working with traceability link issues; again from the perspective of the user and the usability of the tool. This leads us to the following research questions:

1) *How does explicit representation of traceability consistency issues affect issue resolution?*
2) *Which benefits do automated fixes provide to resolve consistency issues in traceability links?*

IV. Overview of the Research Method

To answer our questions we performed a design science research (DSR) study[11]. This type of study consists of the the following activities:

1. Problem identification and motivation: We identified the problem and motivation in the Purpose, Section I.
2. Objectives of a solution: The research goal and questions, Section III, give our objectives.
3. Design and development: This consists of requirement elicitation and software implementation.
4. Demonstration: Experiment (including pilot).
5. Evaluation: Data analysis.
6. Communication: The findings are communicated via this thesis.

Our study consisted of two distinct phases. Each phase contained stages 2 to 5 of the DSR list. Additionally, phase 1

contained a competence test that allowed us to assess our participants capabilities with Eclipse prior to the experiment. The experiments were conducted in a quiet, private room within the school library to ensure everyone had the same conditions.

Throughout the test and experiments we made informal observations on participant behaviour. While we cannot make any claims based on our opinions we have included them in the results and discussion sections for completeness.

We recruited 18 students to take part in our experiments. All the participants were part of the third year of the Software Engineering and Management (SEM) program at Gothenburg University and had experience of working with Eclipse. The program is taught in English and all students are expected to be competent with the language.

A. Phase 1

This section provides a detailed description of the first phase of our research. Section IV-A1 describes the requirements elicitation that was done with rt-labs before any software development was done. Section IV-A2 discusses the implementation of Dash 1. Section IV-A3 gives the details of the test that was designed to assess the skills of the participants within the Eclipse framework. Section IV-A4 covers the first experiment, where we investigated the effects of explicitly representing traceability issues to the user. Section IV-A5 describes the interviews that were done after each experiment. Section IV-A6 explains how the competence tests were graded and Section IV-A7 explains how the experiments were analysed.

1) Requirements Elicitation: The purpose of the elicitation was to get information from rt-labs with regards to the requirements for Dash 1. We decided to work with rt-labs because they were a partner in the creation of Capra and had experience with the tool. We organised a one hour meeting with two members of rt-labs and one of our supervisors. The meeting was held at the rt-labs offices in Gothenburg.

We spent time with Capra prior to the meeting in order to understand the capabilities and limitations of the software and to enable the writing of questions that would be relevant. The main focus of the questions was how to represent the issues and to define what an issue was; this would help us answer our first research question. Our secondary focus, for the second research question, was the definition and use of automated fixes. Questions were written in advance so that a semi-structured interview could take place. The order was not pre-defined but care was taken during the interview to ensure all the different areas would be covered. The interview was recorded with the permission of all present; one of the authors took notes while the other asked the questions. The questions can be found in appendix A.

2) Dash Version 1 Implementation: In order to gather data on our first research question we needed to implement version 1 of Dash. The requirements from rt-labs, presented in Section V-A, were used to motivate the design. Specifically, Dash 1 would notify the user when a traceability link was broken and would apply a problem marker to the offending resource. The notification itself would appear in the problem view and gave the user more information on the resources that caused the issue. A rename warning would display the original and new file names, a move issue would give the original and new locations and a delete would say which file had been deleted and from where.

3) Eclipse Competence Test: In order to perform our study we needed a control group and an experimental group to compare it with. The main purpose of this competence test was to give us an idea of the ability of each student when using the Eclipse IDE and EMF. The score each person got on the test allowed us to create two balanced groups. We wanted the test to give us a good spread of results; not so hard that the less skilled users would give up, but not so easy that everyone got the same high score. We ran two pilot tests for feedback and sample data as the first prompted us to change the design.

The second objective of the test was to teach the participants the necessary skills needed for the experiment. We did not want a lack of basic knowledge to affect our results. For example, the test involved creating a class, if they failed this part we would show how it was done after the test. The test was designed to only include tasks that should have been familiar to the students. Some actions that were included in the test were actions that are possible to do in Capra as well, such as renaming, deleting and moving resources.

The test was split into two parts; the first involved modifying a UML model of a hockey team, using the Eclipse modelling plug-in, Papyrus[4]. The second part used a simple Java project with two classes and an error of a class declaration when no class existed. Participants had to understand and be able to change perspectives (a perspective controls how the interface looks, which views are open and so on) in Eclipse and locate and open the Problem view, which was one of the main features that Dash 1 would utilise for displaying warnings. It should be noted that opening a model in Papyrus would cause the IDE to become unresponsive for a short time but we did not include this as time used by the participant. We tested one participant at a time and used the same computer so that even with the Papyrus problem everyone would get a similar experience.

The specific tasks were as follows:

1. Open the Papyrus perspective
2. Open the model from the root element
3. Add attributes to the person class
4. Create generalisations
5. Delete unnecessary attributes
6. Rename the diagram
7. Save and close the model project
8. Open the Java project
9. Locate the error using the problem view
10. Fix the error using the correct quick fix
11. Rename a class
12. Save the project

[4]https://eclipse.org/papyrus/

A screen recorder was running in the background (which the participants were informed about before the test began) that was used for capturing all the actions so that each video could be discussed and graded accurately afterwards.

4) Experiment One: The purpose of the first experiment was to assess how the representation of traceability issues affects their resolution. The control group used the original, unmodified version of Capra and the experimental group had Dash 1 with warnings, as described in Section IV-A2.

The experiment workspace consisted of a Java project with 5 packages and 26 classes spread between them. There were 13 traceability links created between the classes and 26 artefact wrappers in the container. We created 9 issues so that the artefact wrapper container and the trace model would no longer match the project. Dash 1 displayed the 9 issues as warnings in the problem view while Capra gave no indication that problems existed.

There were 3 different types of issues: rename, move and delete. The rename issue was caused by changing the name of a resource in the project, making the artefact wrapper incorrect. Move issues were caused by a project resource being relocated and the wrapper URI being incorrect. The delete issue was created by removing a resource from the project so that the wrapper would refer to something that no longer existed.

For the purposes of the experiment a rename was fixed by editing the name and URI property of the relevant wrapper. A move was fixed by updating the URI property of wrapper. For the delete issue the requirement was to recreate the missing class. Each fix had to be done manually.

Participants were given a handout introducing them to the Capra workspace, the subject of traceability and terminology used in the experiment. Additionally, the document contained a guided practice for the participants to perform on a simple workspace. The practice taught them how to create traceability links, how issues were created and how to fix them. They were allowed to refer to the document during the experiment.

The participants were not made aware that there were two versions of the tool and the ones with Dash 1 were not given additional instructions, although the problem view was open for the experimental group so that the warnings could be seen.

5) Participant Interview: When each experiment was completed we conducted a short interview to discover how confident people felt with the different tasks and to get feedback on the tool. Each participant was interviewed separately to avoid bias from hearing the other person's answers. The specific questions were: On a scale of 1 to 5, with 5 being the highest:

1. How confident are you with fixing rename issues?
2. How confident are you with fixing delete issues?
3. How confident are you with fixing moved issues?
4. How confident are you that you found all the issues?

To get feedback we asked:

1. How would you improve the process of finding issues?
2. How would you improve the process of fixing issues?

To help ensure that the experiment would give us the data we were looking for we first conducted a pilot test with two of the participants. Their selection was based on the scores from the competence test. We wanted people with average to above average scores as it suggested that they were comfortable with Eclipse yet still able to make mistakes. This way we could consider any problems to be related to the task and not the IDE.

6) Eclipse Test Analysis: Each task that was given could give the participants a score from 0 to 3 points. A score of 3 points would be given when a task was solved in one step and within a reasonable time; meaning that the participant showed awareness of the environment and completed the task in one take without having to redo anything. Based on the pilot test we observed that mistakes could occur, for example, accidentally hovering and selecting the wrong menu option would not withdraw any points from the participant if they corrected the mistake immediately. If they would realise the mistake later on, this would take away a point. The score of 2 was given when participants completed the task correctly but took more than 1 attempt. For example, understanding a rename task but not being aware of which menu held the option. For this step there also was a time limit of 1 minute and a maximum of 3 tries to complete the task. If the behaviour of the participant showed that they were lost and looking through every option and managed to get the task right at more than 3 attempts, then this score went down to 1 point. Lastly, the score of 0 was given when the tester showed that they did not know what they were doing at all and taking far too much time while randomly guessing and clicking; 0 points is also given when task is incorrect. The exception to these rules would be if the participant had not done the MDD course and had no experience with Papyrus. If they were unable to open the perspective or the model within 1 minute they would receive help but only get 1 point.

One member of experimental group dropped out at the last minute. The remaining 17 participants were arranged in groups according to their scores. The person with top score was the first one into the control group. Second highest was put in the experimental group, the third to the control group and so on until all the participants had a group.

7) Experiment 1 Analysis: To analyse the experiment we extracted the following data from the recordings:

1. The time of the first fix
2. The total time taken to complete
3. Number of issues found
4. Number of issues fixed correctly

The time for the first fix was recorded and used as the start point because we expected some participants to struggle to get started; either through a lack of understanding of the tool, the task or both. This opinion was based on observations made during the experiments. We compared the total time (from first fix) of a participant from the control group with a person of roughly equal skill (based on their competence test score) from the experimental group. The next step of the analysis was to compare the average number of faults found and fixed between the 2 groups, as well as the average time taken to complete.

For the interviews we tabulated the quantitative data that described user confidence and took averages of both groups to compare. For the feedback we coded the responses from both groups into separate tables and recorded the number of requests for each improvement.

B. Phase 2

Phase 2 follows the same structure as phase 1. Section IV-B1 explains the requirements for Dash 2. Section IV-B2 describes how the requirements are interpreted. Section IV-B3 covers the second experiment where we investigate automated fixes. Section IV-B4 describes how we obtained data on and control. Section IV-B5 explains how we analysed the recording of the second experiments.

1) Requirements from Experiment 1: The results of the phase 1 interview analysis, presented in Section V-B, were compared to the original requirements from rt-labs. The most requested improvements that also appeared on the rt-labs list were chosen to be implemented.

2) Dash Version 2 Implementation: The feedback from the interviews done during experiment 1, presented in V-B, as well as the original requirements, was used to motivate the next step of the implementation. Quick fixes and automated fixes were the most requested improvements; while warnings also scored highly this feature already existed in Dash 1. Therefore, the purpose of this phase was to add some automation to issue resolution, via quick fixes, to Dash 2. A quick fix in Eclipse can be accessed through right clicking on a warning in the problem view. The purpose of a quick fix is to offer the user options for correcting the issue, with the fix itself being automated. For example, a quick fix for the rename issue would offer to change the properties of the affected wrapper.

3) Experiment 2: The purpose of the second experiment was to answer the second research question by investigating the benefits of automated fixes. Additionally, we wanted to investigate the concepts of trust and control when using the tool. Control refers to how much a person can determine the behaviour of the tool or how much they can affect what happens when fixing issues. Trust is used in terms of the user and the tool when using manual and quick fixes. For example, do they trust themselves more or less depending on the version of Dash. Unfortunately, one of the experimental group had failed to attend previous experiment. Because of this we were reduced to 16 participants, the control group was now 9 people with 7 in the experimental group. In this experiment the control group would use the first version of Dash, with notifications, and the experimental group using Dash version 2 that had quick fixes.

For the second experiment we created a new project for the participants to work in. This was done to get the experiment to feel like part of a normal work day for someone using Dash. The new project consisted of several folders designed to simulate a requirements engineering project.

- Activity Diagrams - contained 4 png image files, one being misplaced.
- Class Diagrams - contained 4 png files.
- Sequence Diagrams - contained 5 png files.
- Domain Model - contains 2 png files representing parts of the projects domain model.
- Source code - a hotel management project with 2 folders, 1 for the source code and 1 for test classes. The source code folder had 4 sub-folders with numerous classes in each.
- Requirements - 28 requirements for the project, plus 1 misplaced file.
- Use Cases - had 25 objects where 4 were misplaced.

Unlike the first experiment there were no issues in place at the start; the trace model and artefact wrapper container matched the project. We had wanted to use Papyrus again for the different diagrams but had problems with performance. Opening a single model during the competence test had proved troublesome; in some cases it took up to a minute to open and Eclipse would be unresponsive while waiting. While a clean installation of Eclipse might have resolved this there was no guarantee of the performance being the same for each person. Even though there was no reason for them to open the models that did not mean they would not do it. Instead we opted for png image files to represent the diagrams instead. This had no bearing on the outcome of the experiment as the information contained in the files was not important.

Traceability links were created between some of the requirements and their respective Java class, including all the files in between. For example, requirement to use case to class diagram to test case and Java class. Additionally, some files were placed in the wrong locations for the user to correct during the experiment. It's important to note that not all files were linked and therefore would not cause any issues when modified. This was done to try and force the participant to think about what they were doing and react to the tool instead of assuming that every change would create an issue to be fixed.

The possible issues were the same as experiment 1: rename, move or delete. However, the fix for a delete issue was changed. The previous fix was to recreate the missing resource. Manual users would instead now have to set all properties of the relevant wrapper to NULL when deleting a resource from the project that caused an issue. This new fix would also be performed automatically by the quick fix.

The participants were required to follow a list of tasks that would simulate normal activity for someone working with the tool. The possible tasks were:

- Create a traceability link
- Rename a file
- Delete a file
- Move a file
- Create a folder

If a task created an issue the user was expected to fix it before moving to the next task. The full list of tasks can be found in appendix B.

As in the first experiment, a handout was written for each group. All participants had now experienced Capra so the

introduction to the tool was not included. For the same reason we did not include a compulsory practice session. Instead, the handout covered the possible tasks and how to perform them. If they had version 2 of Dash the handout told them how to use the quick fixes, if they had version 1 they were told how to perform the new delete fix. The participants had the opportunity to ask questions about the other fixes if they needed a reminder. We expected the participants to remember how to perform rename and move fixes and this opinion was supported by the feedback from the pilot test.

4) Participant Interviews: We observed during the first experiment that some users that had the notifications were hesitant to say that they had finished when they had fixed the last known issue, the opposite was true for participants that had to check each wrapper by hand; when they checked the last wrapper they knew were done. We thought it was possible that there was a lack of trust. Lack of trust in the tool could come, for example, from the tool being new or a prototype. Lack of trust in the people running the experiment could come from the idea that we would try and catch people out. We also wanted to investigate whether or not the level of control that people felt over a tool or operation changed with automated features. We decided to investigate this further by including it the interviews.

The interviews were conducted with both participants in the room where possible. This was done because we wanted the second part to be more of a discussion. The first question, on a scale of 1 to 5, with 5 being the highest, was about the user's confidence that the model correctly matched the project, i.e. were all the tasks done correctly and the right fixes implemented when necessary. The second question, on a same scale, was about how aware they were of the effects of the changes they made in the project. For example if an object was renamed, were they aware of the wrapper container being out of date. Lastly, on the same scale, how in control did the participants feel during the experiment, and if that would change if they had the other version of Dash. The second part of the interview introduced the concept of trust to the participants. We wanted to know how their opinion of trust would change with the different versions of Dash.

A pilot was done to test the methodology of the second experiment. Due to the quality of feedback from the first pilot we decided to use the same people for the second pilot test.

5) Experiment 2 Analysis: To analyse the experiment we recorded the time taken to complete all the tasks and gave a score for how well the task was done. A participant could get 1 point for doing the task correctly, half a point for a small mistake, such as having the source and target of a traceability link the wrong way round, and 0 for being incorrect or skipping the task. In addition to this any issue that was generated would be treated as a new task, with the same scoring for the user when fixing the problem. The maximum score available was 38 with the maximum time allowed being 40 minutes.

V. Results

This section presents the results for the study. For the test and experiment subsections we also give our initial interpretations of the results.

A. rt-lab Requirements

The requirements from rt-labs were, in no particular order, as follows:

1. Dash should automatically notify the user when a link becomes invalid through one or more artefacts being moved, deleted or renamed.
2. The user should approve of fixes before the system performs them.
3. Markers should be shown on the offending artefact where possible.
4. Quick fixes should suggest possible resolutions to problems.
5. It should be possible to fix issues in bulk. E.g. fix all links with renamed artefacts.
6. Notifications (markers) should appear in the problem view and project explorer view.
7. It should be possible to filter by issue type in the problem view.
8. Issues should be checked for when a save is performed.
9. Broken links should be shown in red on the graphical representation.
10. Problem markers should persist between sessions.
11. User should be able to run the traceability check on demand.
12. Options should exist to create a rule set that would control what the tool checks for, how strict the check is (e.g. should all artefacts have a traceability link), ability to toggle settings such as warnings to show.

B. Eclipse Test and Experiment 1 Results

1) Pilot Test: The feedback for the pilot test was positive with only adjustments to the handout needed. The 2 participants are included in the overall results.

2) Eclipse Test and Experiment 1 Results: Table I shows the results of the test and the experiment. The score for each participant is in column 3 and the table is ordered from highest score to lowest, while alternating between A, control group, and B, the experimental group. Participants 3 and 10 are repeated to allow us to compare people of similar test scores. DNF indicates the participants did not finish within the 10 minute limit. The maximum number of issues that could be found was 9. The rows are the two participants that are being compared. We can see from the table that the worst performers from the competence test are the worst in the experiment. The same can be suggested for the top scorers but it is not so conclusive here.

Table II shows a summary of Table I with the averages for each group for time to complete, fixes found, correct fixes and the average test score from a maximum of 35 points. Participants 3 and 8 are not included in the averages for time as they did not finish. We can see that very little separates

TABLE I
TEST AND EXPERIMENT 1 RESULTS

ID	Group	Test Score	Time (sec)	Issues Found	Issues Fixed
18	A	35	272	9	9
4	B	33	360	9	7
17	A	33	444	9	9
6	B	32	219	9	9
14	A	30	416	9	7
15	B	30	313	9	9
5	A	29	360	9	7
10	B	29	353	9	7
16	A	29	259	9	9
10	B	29	353	9	7
1	A	26	356	8	8
13	B	20	410	9	8
7	A	18	376	9	9
12	B	18	250	9	8
9	A	16	353	9	9
3	B	15	DNF	5	5
8	A	10	DNF	5	0
3	B	15	DNF	5	5

TABLE II
EXPERIMENT 1: GROUP COMPARISON

	Control	Experimental
Average fixes found	8	8.4
Average correct fixes	7.4	7.6
Average time to complete (sec)	357	295
Average test score (max 35)	25.1	25.3

the 2 groups in terms of finding and fixing issues, although the control group averages just over a minute slower for completing the task.

Table III shows how confident the user was in the different types of task, plus how confident they were that they found all the issues in the system. The scale is from 1 to 5, with 5 being the most confident. Table IV compares the averages from both groups. They show little difference between the groups for fixing the issues but some individuals have interesting variance in confidence between the different fixes. What is clear is that

TABLE III
EXPERIMENT 1: USER CONFIDENCE

ID	Group	Rename	Delete	Moving	Found all
18	A	5	5	5	5
4	B	5	5	4	5
17	A	5	4	3	5
6	B	5	5	5	5
14	A	4	2	2	3
15	B	5	5	5	5
5	A	5	4	4	4
10	B	4	4	4	5
16	A	5	5	5	4
1	A	5	5	5	3
13	B	4	4	4	5
7	A	5	5	5	4
12	B	3	5	3	5
9	A	5	4	4	5
3	B	3	4	2	5
8	A	3	5	5	5

TABLE IV
EXPERIMENT 1: AVERAGE USER CONFIDENCE

	Control	Experimental
Rename	4.67	4.14
Delete	4.33	4.57
Move	4.22	3.86
Found all issues	4.11	5

the experimental group had complete confidence in the tool to show all the issues correctly.

Table V shows the interview coding for the control group. The first row shows the requested improvements from the users of Capra. What follows describes each code.

- Warnings: This is the functionality that was implemented in version one of Dash where warnings show in the problem view.
- Marker on wrapper: Users felt that having a marker on the wrapper with the issue would aid in fixing the issue.
- Automated fixes: Users wanted the issues to be handled and fixed automatically with no input from the user.
- Quick fixes: Quick fixes were requested as a way of being able to select the type of fix to be done with the fix itself being automated, allowing the user to maintain some control.
- Sort Wrappers: Being able to sort the wrappers, e.g. alphabetically, would make it easier to manually traverse the list when searching.
- Fix multiple issues: It was suggested to have multiple similar issues fixed with a single command, e.g. fix all rename issues.
- Improve wrapper UI: Participants wanted an easier way to work with the wrapper container and changes with the properties of the wrappers.
- Colour Notifications: We had participants asking for some kind of colour change when an link is broken and when it is fixed.

Table VI contains the results of the interviews with participants from the experimental group. Some of the requested features have already been described, the rest are below:

- Separate views: This refers the default layout in Capra where the problem and properties views occupy the same space, forcing the user to either change the layout or switch back and forth between the two views.
- Notify when missing: A specific notification when an object is missing from the project explorer. Could be a warning in the problem view, a pop up or a marker on the parent folder of the missing resource.
- Open wrapper on click: This feature would allow the user to double click on the warning in the problem view and have the problem artefact wrapper opened in the main window.

The most requested improvement is quick fixes (10 requests), closely followed by automated fixes and warnings (7 requests respectively). These, and many of the other improvements asked for, match the original requirements from rt-labs.

TABLE V
EXPERIMENT 1: CONTROL GROUP INTERVIEW CODING

ID	Warnings	Marker on wrapper	Automated fixes	Quick fixes	Sort wrappers	Fix multiple issues	Improve wrapper UI	Colour notifications
1	✓	✓	✓					
5	✓						✓	
7	✓	✓		✓				
8			✓			✓		✓
9	✓			✓				
14	✓	✓		✓				
16			✓		✓			
17	✓			✓		✓		
18	✓			✓			✓	
Total	7	3	3	5	1	2	2	1

TABLE VI
EXPERIMENT1: EXPERIMENTAL GROUP INTERVIEW CODING

ID	Separate views	Automated fixes	Quick fixes	Sort warnings	Fix multiple issues	Notify when missing	Open wrapper on click	Marker on wrapper
3	✓	✓	✓					
4			✓	✓	✓	✓		
6		✓						
10		✓	✓				✓	
12				✓				
13			✓					✓
15		✓	✓					
Total	1	4	5	2	1	1	1	1

TABLE VII
EXPERIMENT 2: SCORES AND TIMINGS

ID	Version	Score	Time (Min)
18	A	33.5	20.24
4	A	38	33.21
17	A	31.5	24.59
6	B	36	9.2
14	A	23.5	29.3
15	B	38	15.38
5	A	34	18.46
10	B	34.5	22.56
16	A	26.5	15.15
1	A	24.5	37.09
13	B	32.5	15.57
7	A	33.5	21.53
12	B	35.5	19.17
9	A	32.5	37.47
3	B	24	14.36
8	A	15.5	22.14

TABLE VIII
EXPERIMENT 2: SCORES AND TIMINGS COMPARISON

	Control	Experimental
Average score	29.3	33.4
Average time to complete (min)	25.9	16.04

C. Experiment 2 Results

1) Pilot Test: The pilot test feedback changed the interview slightly. The original interview had the question about trust on a scale of 1 to 5. However, trust was a difficult concept to communicate and the interview became more of a discussion. We decided instead to make the final question about trust and control and just document the comments instead.

2) Experiment 2 Results: Table VII shows the participants in the order of their competence test scores, their score for the experiment and the total time it took them to complete the experiment.

It should be noted that a mistake was made with participant 4 as they were given the wrong handout and were not aware that quick fixes were enabled. For this reason they have been switched to the control group. This made it 10 for control and 6 for the experimental group.

Table VIII compares the 2 groups. The scores are close but there is a significant difference between the times to complete of almost 9 minutes.

Table IX shows the quantitative results from the interviews, where the answers were on a scale between 1 and 5, 5 being the highest. Table X shows the averages of the two tables together. While the level of confidence is close the control group had a higher awareness of the effects of their actions. The control group also believe they would have had more control with quick fixes. However, the experimental group thought that they would have more control with manual fixes.

Table XI shows the coded qualitative results from the interviews about control and trust. The codes are defined as follows for quick fixes:

- More trust when new: People would trust the tool more when they were new to it.

TABLE IX
EXPERIMENT 2:USER CONFIDENCE AND CONTROL

ID	Confidence that model matches project	Awareness of effects	Level of control with manual	Level of control with quick fix
Control Group				
1	3	2	4	5
4	4	4	4	5
5	4	5	3	4
7	4	5	4	5
8	5	5	5	3
9	4	3	4	5
14	2	5	4	4
16	2	5	3	4
17	3	4	4	5
18	4	4	5	4
Experimental Group				
3	1	1	4	3
6	5	4	5	4
10	4	4	4	5
12	3	3	4	5
13	5	5	5	5
15	5	4	5	3

TABLE X
EXPERIMENT 2: USER CONFIDENCE AND CONTROL COMPARISON

	Control	Experiment
Confidence that model matches project	3.5	3.8
Awareness of effects	4.2	3.5
Level of control with manual	4	4.5
Level of control with quick fix	4.4	4.2

- Less control when experienced: Control would increase with experience.
- Trust in tool to work: People would trust the tool to correctly fix issues.
- More control: Control increases when using quick fixes.
- Less control: Control is less due to the automation.
- Trust once tested: The tool would only be trusted once they had seen that the fix was correct.

For manual fixes we have:

- More trust in self: Trust increases with experience.
- Less control when new: Control increases the more they use the tool.
- Less trust with user error: They believe they will make mistakes.
- More control: More control when manually fixing.
- Trusts self: Trusts self to fix correctly.
- Less control: Less control when manually fixing issues.
- Control same for both
- Control increases over time

The most notable results here are that 12 people thought that quick fixes could be trusted to work correctly. Although 8 thought that it meant less control 6 people thought the opposite. User error was an expected problem for 9 people but 7 thought they could trust themselves to fix issues correctly. As with the quick fixes, the amount of control people felt they have was split with 7 saying more and 4 less.

3) Observations on Participants: During the competence test we observed that people would select the first option in a quick fix dialogue, suggesting that they either weren't reading the options or didn't know which option was correct. Some users would read the task description then perform the task differently to the way described. For example, when told to fix an error with a quick fix some users found other ways to remove the error.

In the experiments we saw a competitive element emerge that was not present during the competence test. The participants were generally kept in their thesis pairing and we could see they wanted to outperform each other. When one finished the other could become frustrated, people who tried to go too fast would make mistakes; for example, 2 users managed to create new warnings when rushing.

Despite the existence of the handout that could be used at any time, people still made mistakes when correcting issues during the experiments. Confidence when fixing issues was also erratic for some, even with precise instructions.

The second experiment had 3 participants who did not attempt to fix any of the issues that were created. When asked they told us that they had not read the instructions correctly. They were not alone in this behaviour; some participants missed tasks or performed a task in a way that suggested they had not read the description correctly, such as making a traceability link between the wrong source and target.

VI. DISCUSSION

This section discusses the results of the 2 phases in terms of the research questions, participant behaviour, the validity of the work and the next steps for both the tool and any future study that could build on this work.

A. Phase 1

The competence test results showed us that the participants we had chosen had a broad range of ability. This was beneficial in terms of creating balanced groups but it could also be said that professionals who use Eclipse daily would not have struggled and would make better participants for testing a plug-in.

1) How does explicit representation of traceability consistency issues affect issue resolution?: The results for experiment one are inconclusive for demonstrating the advantages of the Dash 1 extension. When comparing participants of similar skill levels (from the results of the competence test) we found that there was no clear advantage with either Capra or Dash. Users of Dash had a higher level of confidence that they had found all the issues although they also made more errors when fixing issues.

In terms of finding all the issues, 5 members of the control group showed a lack of confidence that they had found all the issues. This is despite the fact that only 1, other than those that did not finish, failed to find all the problems. This suggests that without any feedback from the tool users are unwilling to trust themselves completely. This is backed up by the fact

TABLE XI
EXPERIMENT 2: INTERVIEW CODING

	Quick Fix						Manual Fix							
ID	More trust when new	Less control when experienced	Trust in tool to work	More control	Less control	Trust once tested	More trust in self	Less control when new	Less trust with user error	More control	Trusts self	Less control	Control same for both	Control increases over time
1	✓	✓			✓			✓		✓	✓			
3			✓						✓					
4			✓	✓				✓	✓			✓		
5			✓	✓				✓	✓					
6			✓		✓					✓	✓			
7			✓		✓					✓	✓			
8	✓				✓					✓	✓			
9					✓	✓			✓	✓				
10					✓	✓	✓			✓	✓			
12			✓		✓				✓	✓				
13			✓	✓					✓				✓	
14			✓						✓				✓	
15			✓	✓					✓			✓		
16			✓	✓					✓			✓		
17			✓	✓							✓	✓		
18			✓		✓						✓			✓
Total	2	1	12	6	8	2	1	3	9	7	7	4	2	1

that 7 of the 9 people wanted warnings added to the tool to highlight when issues had occurred.

Generally, the users of Capra would check an artefact wrapper against the project, fix if an issue was found, and then take the next wrapper. While this approach was competitive in a small project against Dash we believe that it scales poorly. Checking for an unknown number issues in 26 wrappers would clearly take less time than checking through 300 wrappers. Comparing this to Dash, where the name of the artefact wrapper and the total number of issues is known, users should be significantly faster; the only difference being the increase in time it would take to locate the effected wrapper.

Observations made during the experiments showed that the UI design was not optimal for Dash users. The properties and problem views default to occupying the same space in Eclipse and users were forced to switch between the two; first to understanding the issue and then to edit the properties. It is important to note, however, that there was no restriction in place to prevent them from moving either view. Capra users had all the information they needed available without needing to switch views.

B. Phase 2

1) Which benefits do automated fixes provide to resolve consistency issues in traceability links?: The most obvious benefit shown is that automated fixes speed up the process of resolving issues with an average of about 9 minutes separating the 2 groups. While this appears to be significant we should note that the experimental group was smaller by 4 people. The size of the population, at 16, is not big enough to statistically prove this benefit.

Only participant 8 of the control group had complete confidence that they had performed the tasks correctly and the models matched the project. However, they scored low on the competence test (10 points) and in both experiments (DNF and 15.5 points). This suggests that the confidence was in themselves and not related to the tool as they were oblivious of their poor performances.

In the experimental group 3 out of 6 had full confidence. Participant 3 was aware after they had finished that the tasks had not been completed correctly and gave a confidence of 1 out of 5. Number 10 was confident but was not sure they could trust the tool. Number 12 didn't trust themselves to do it right regardless of the performance of the tool.

The observations made during the experiments suggest that human error can occur very easily, people would misread or ignore instructions, or not show enough care when fixing an issue. We believe that while the automated component of quick fixes would not remove human error it does ensure that the trace model stays up to date.

2) Trust and Control: The results of the interviews suggest that quick fixes give a high level of trust in the system to work correctly. Half the participants felt that control went down as they were no longer making the changes directly. For manual fixes it can be said that over half of the participants did not trust themselves to never make mistakes when fixing issues; a side effect of having more control with manual fixes is more actions that can go wrong. While 7 people said that they trusted themselves to not make mistakes it should be pointed out that the experiment was only 35 minutes of work at most and only 2 of the 16 got the maximum score of 38 points.

C. Participant Behaviour

This section discusses the observations reported in Section V-C3. The behaviour with quick fix options could be explained by a lack of user knowledge or experience with Eclipse. When they selected the wrong option most people went back and tried a different solution.

The competitive nature was missing during the competence test as we performed these with individuals instead of pairs. It is possible that people were more concerned with 'winning' against their partner instead of focusing fully on the task they had.

There are several possible reasons for mistakes being made even with a handout available. These reasons can also apply to those that did not attempt to fix anything. First is the pressure from being part of an experiment and not wanting to fail. Second is that no one wants to be last to finish; if one person has finished reading the other might skip to the end to avoid embarrassment. If they had known that they could have different versions of the experiment and were not expected to finish at the same time this could have reduced this pressure.

D. Threats to Validity

Our threats to validity are based on those laid out by Easterbrook [12].

1) Construct Validity: Our Eclipse competence test was done to avoid getting only participants with extreme scores. As table I shows, Section V-B, we had a good range of abilities. While this does not have validity issues it should be noted that it was designed to separate students with limited experience of Eclipse. This design would not work for industry professionals with much greater experience; we would expect them to generally have high scores.

2) Internal Validity: We lost 2 participants during the study. The first had still not done the test shortly before the experiments were due to start. The second did the test but missed the experiments due to work commitments. While we could have tried to recruit a larger number of people we felt (and were proved correct) that our already tight schedule would not allow for this. While this does affect our sample size it does not change our findings as it was never possible to do statistical analysis with just 18 people.

While we were careful to prevent communication during the experiments there was nothing we could do to stop people from the different groups talking afterwards. The handouts and the tool were not available out of the experiment room.

To prevent the control group from feeling devalued we did not inform people which group they were in. However, we did have people from each group doing the experiments at the same time and the control participant, with what has been shown to be the slower method, could be affected by the experimental participant finishing more quickly.

As we knew all the participants it is possible that we behaved differently depending on who was doing the experiment. We attempted to reduce this threat by using handouts to instruct the participants instead of us risking giving varying instructions verbally. It also guaranteed that everyone had exactly the same knowledge of our tool and experiment. Screen recordings were used to avoid us distracting the participants, either with our reactions or just our presence.

Rather than avoid people learning how to respond during the experiments we decided to encourage the learning effect. Our competence test taught them the skills needed for the first experiment while also assessing them. We made sure that everyone had the same information and experience during the first experiment with regards to fixing issues. In the second experiment we designed the tasks around everything they had done with us before.

3) External Validity: Our initial idea was to recruit volunteers from the 3 years of the SEM program. This was done via social media and we only received one valid response. This forced us to approach students in our own year that we were familiar with and ask them directly. This produced a much better response and we were careful to avoid asking only students with the best grades. The downside here is that we could be biased to certain people and vice versa. We believe that the design of the test, experiments and interviews prevented this from negatively affecting the study; it's possible that the participants knowing us made them more willing to be a productive part of the study. However, the use of students prevent the findings from being applied to any other group, for example, people in the software industry.

VII. Conclusion and Future Work

While there is no clear difference with a small project, it is reasonable to assume that with a large enough project the users with notifications will be faster at issue resolution. This is due to the fact that the amount of work involved when you have notifications does not scale with project size; fixing 9 issues in a 300 wrapper project is virtually the same amount of work as fixing 9 in 100 wrappers for a user with notifications. With no issue notification the user always has to check the whole project. It can also be said that users with no notifications lacked the confidence in themselves to maintain an up to date model.

Automated fixes have been demonstrated to speed up resolution of consistency issues. Having both notifications and quick fixes can reduce human error and help maintain traceability in large scale projects, while reducing time lost checking and correcting traceability link consistency. Users trust quick fixes more than themselves to keep a model correctly up to date with a project but can feel a loss of control over doing it manually. It is clear that even when users believe they are right the majority are still able to make mistakes.

A. Dash 3: The Next Steps

While the warnings worked for representing issues we did observe people either not noticing or ignoring them. A warning in Eclipse does not prevent the code from compiling and they can even be deleted. As Capra is being developed for an industry with strict rules regarding traceability we recommend

that the warnings are upgraded to errors to prevent the user from continuing to work without fixing the issue.

Implementing the visual representation of the trace model as a graph is a requirement that rt-labs wanted but never made it in. If this were added, including showing issues on the traceability links, it would give users a graphical alternative to text for working with the trace model.

B. Further Study

The work done in this study could be continued by testing with industry; starting with rt-labs. As the results of the first experiment were inconclusive we believe that it could be repeated with a group large enough to perform statistical analysis on. Additionally, the size of the test should be increased to contain enough wrappers to demonstrate a clear difference between the control and experimental groups.

If the next steps for Dash 3 were implemented then it would be worth experimenting on the benefits of graphically representing issues on a trace model. For example, investigating the benefits of graphical representation compared to textual.

VIII. Reflections

As this was the first research study we have carried out we think it is important to reflect on what we learned and what we would do differently if we repeated the study. We would recruit participants who were more proficient with Eclipse. Low scores from the competence test gave poor results in the experiments. We would attempt to remove the issues that arose from having 2 participants do the experiment at the same time. We could, for example, do them all individually or have only the people from the same group. Human error when reading instructions was unexpected. We could improve this by having both the handout and verbal instructions.

Acknowledgement

We would like to thank our supervisors, Jan-Philipp Steghöfer and Salome Maro. rt-labs for making time for us and helping with the initial tool design. Finally, we would like to thank the willing participants for their time, patience and understanding throughout their final term.

References

[1] J. Radatz, A. Geraci, and F. Katki, "IEEE standard glossary of software engineering terminology," *IEEE Std*, vol. 610121990, no. 121990, p. 3, 1990.

[2] S. Winkler and J. Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Software and Systems Modeling (SoSyM)*, vol. 9, no. 4, pp. 529–565, 2010.

[3] H. U. Asuncion, F. François, and R. N. Taylor, "An end-to-end industrial software traceability tool," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007, pp. 115–124.

[4] J. Leuser, "Challenges for semi-automatic trace recovery in the automotive domain," in *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE Computer Society, 2009, pp. 31–35.

[5] E. Ben Charrada, A. Koziolek, and M. Glinz, "Identifying outdated requirements based on source code changes," in *Requirements Engineering Conference (RE), 2012 20th IEEE International*. IEEE, 2012, pp. 61–70.

[6] P. Mader and A. Egyed, "Assessing the effect of requirements traceability for software maintenance," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 171–180.

[7] S. Maro, "Improving software traceability in the development of automotive embedded systems - a research abstract," http://ceur-ws.org/Vol-1564/, 2016, [Online; accessed 16-March-2016].

[8] S. Nair, J. L. de la Vara, and S. Sen, "A review of traceability research at the requirements engineering conference re@ 21," in *Requirements Engineering Conference (RE), 2013 21st IEEE International*. IEEE, 2013, pp. 222–229.

[9] I. Santiago, A. Jiménez, J. M. Vara, V. De Castro, V. A. Bollati, and E. Marcos, "Model-driven engineering as a new landscape for traceability management: A systematic literature review," *Information and Software Technology*, vol. 54, no. 12, pp. 1340–1356, 2012.

[10] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements traceability: a systematic review and industry case study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 03, pp. 385–433, 2012.

[11] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[12] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to advanced empirical software engineering*. Springer, 2008, vol. 93.

Appendix

A. rt-labs Interview Questions

The questions were as follows:

General questions:

1. What issues do you want to be notified of?
   - Deletion
   - Moved
   - Created
   - Renamed
2. Contents modified: can it lead to trace problems?
3. What is the order of priority for these issues?
4. What methods do you want to use to notify the user? E.g. pop ups, problem tab, visually on Plantuml (labels, colour changes), markers on resources etc.
5. At what point would you consider notifications to be spam? Think in terms of number of pop ups, clutter on diagrams.
6. When do you want the issue check to run? E.g. on plug-in start, on workspace resource change, both. Menu option to check whole project on demand.
7. What user options would you want? E.g. toggle where notifications are shown.
8. Colour preferences for Plantuml, e.g. link/objects colours changing with status.
9. How much detail in a notification, what type of detail - show more on mouseover, in problem view etc.
10. Should the problem markers be persistent between Eclipse sessions?

Automated Fixes: Our definition: A background operation that updates the workspace when changes are made.

1. Which fixes should happen as background operations?
2. Should we notify the user when automated fixes happen? Include EMF?

Quick Fixes: Our definition: An operation selected by the user from a list suggested by the workspace.

1. What options would you like to see from quick fixes? E.g. Fix all similar problems.
2. What possible fixes are there for each issue?

Advanced:

1. Should there be an option for identifying artefacts without a link?
2. When would the notification happen?

B. Experiment Two task List

1. Create a traceability link between R18 and UC1
2. Create a traceability link between UC1 and F17
3. Create a traceability link b/w UC1 and F28
4. Create a traceability link b/w UC1 and F35
5. Create a traceability link b/w F35 and TestBooking.java
6. Create a traceability link b/w TestBooking.java and UserBooking.java
7. Rename R16 to R16C (C stands for customer)
8. Rename UC2 to UC2_Staff
9. Create a new General Project, call it UseCaseDiagrams
10. Move the three .png files from the UseCases project to the UseCaseDiagrams project
11. R28 - The customer can pay bills with NOK is no longer a valid requirement, remove it
12. Traceability links between ClassDiagrams and PetHotel-Tests have not yet been created, create them. E.g. F35CDBooking.png with TestBooking.java (this one already exists)
13. Rename UC1 to UC1_Customer
14. R23, R24 and R25 are non-functional requirements and will not be in any trace model. Rename them. E.g. R23 to NFR23
15. F2Account.png is not in the right place. Move it to DomainModel project.
16. Requirement R19 is no longer needed. Delete R19
17. Create traceability links between each java test file and its corresponding java controller file.
18. The ”ReadMe” files for requirements and use cases have been placed in the wrong folder, correct this.

C. Capra-Dash User Interface

This section explores the graphical user interface (GUI) used in Capra and Dash. Figure 1 shows the standard Eclipse project explorer. The top folder (WorkspaceTraceModels) is generated by Capra when a traceability link is created. It contains the artefact wrappers and the trace model itself. The other folders are the ones used in experiment 2.

Figure 2 shows the trace model. In this case each traceability link connects 2 artefacts.

Figure 3 shows the properties of the selected traceability link. All links have a source and target artefact.

Figure 4 shows the artefact wrapper container and to the left are the properties of a single artefact wrapper. These are the properties that manual users needed to edit when fixing inconsistency issues.

Figure 5 shows the problem view. In this case there is a single warning notification of an issue giving the user all the

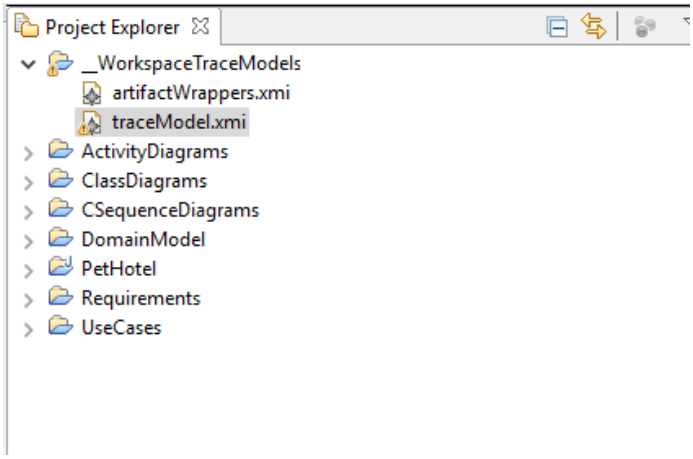

Fig. 1. Project Explorer

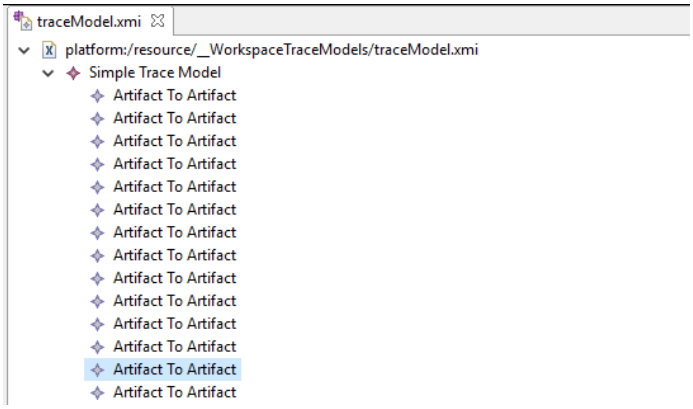

Fig. 2. Trace Model

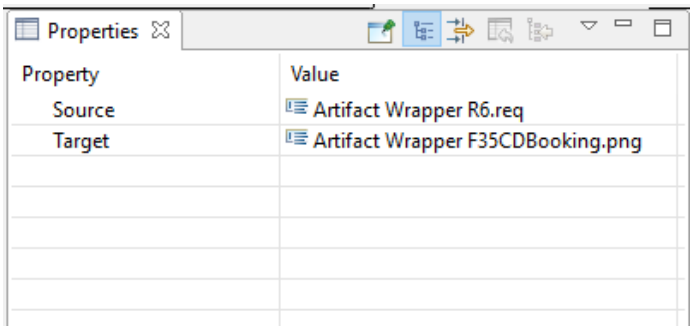

Fig. 3. Properties View - Traceability Link

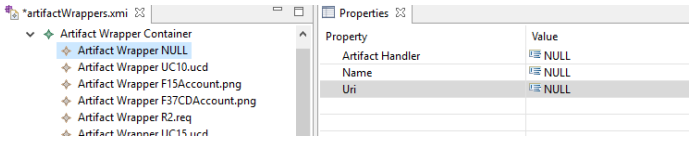

Fig. 4. Properties View - Artefacts and Properties

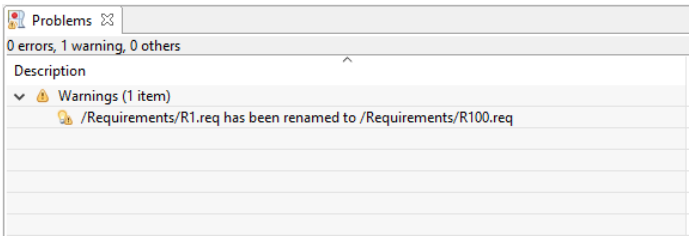

Fig. 5. Problem View

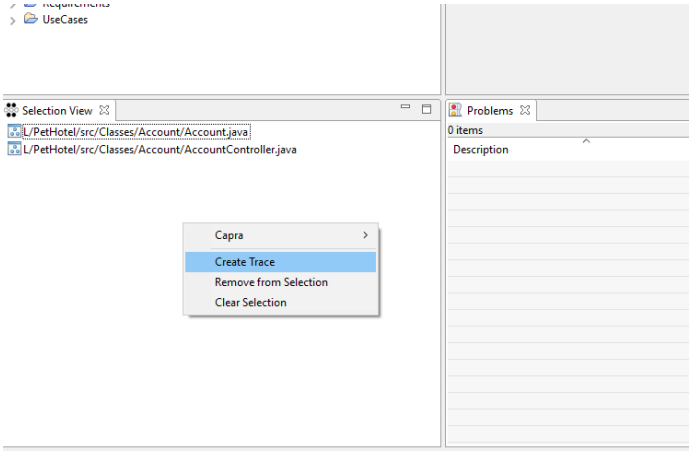

Fig. 6. Selection View

information needed to fix it. For quick fixes the user simply had to right click on the warning and select quick fix from the menu.

Figure 6 contains the selection view being used to create a traceability link between 2 project resources. Users had to drag and drop the resources into the view.

The final image, Figure 7, shows all the views put together as they were for the second experiment. For the first experiment the properties view occupied the same area as the problem view (the Eclipse default setting) and users would switch between the 2 views.

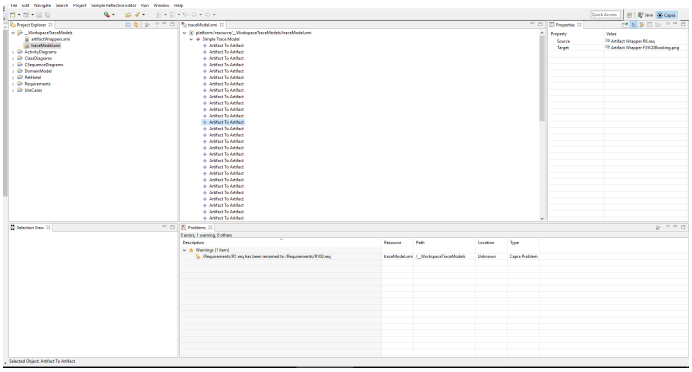

Fig. 7. Capra-Dash User Interface