Answering Software Evolution Questions: An Empirical Evaluation

Lile Hattori, Marco D'Ambros, Michele Lanza

REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

Mircea Lungu

Software Composition Group @ University of Berne, Switzerland

Abstract

Developers often need to find answers to questions regarding the evolution of a system when working on its code base. While their information needs require data analysis pertaining to different repository types, the source code repository has a pivotal role for program comprehension tasks. However, the coarse-grained nature of the data stored by commit-based software configuration management systems often makes it challenging for a developer to search for an answer.

We present Replay, an Eclipse plug-in that allows developers to explore the change history of a system by capturing the changes at a finer granularity level than commits, and by replaying the past changes chronologically inside the integrated development environment, with the source code at hand. We conducted a controlled experiment to empirically assess whether Replay outperforms a baseline (SVN client in Eclipse) on helping developers to answer common questions related to software evolution. The experiment shows that Replay leads to a decrease in completion time with respect to a set of software evolution comprehension tasks.

1 Introduction

When evolving a code base, during software development or software maintenance, developers keep a mental model of the system—an internal working representation of the software under consideration [28]. This individual understanding of the system is constantly being updated by the developer's interactions with the code and the team, and by seeking answers to various questions [26, 2, 6, 15]. These questions span multiple areas [25] such as program comprehension, software evolution, collaborative software development, and program analysis; therefore, they require a variety of information sources (*e.g.,* colleagues, code bases, issue trackers, documentation, communication history), and multiple tools (*e.g.,* [27, 17, 1, 30]) to fulfill them.

Although there are a number of resources (data and tools) available to ease the comprehension of a system and its evolution, the amount of resources actually used by developers is often limited to talking to colleagues and exploring the code.

In an exploratory study [19], LaToza *et al.* report that: 1) almost all teams have a *team historian*, who is the go-to person for questions about the code; 2) most team members subscribe to the check-in messages to keep themselves updated with the code evolution, though many of them expressed dissatisfaction with the lack of detail provided by their teammates when describing the changes in commit messages.

We argue that this *lack of detail* is a fundamental problem for understanding software evolution, *i.e.,* changes made by other developers. The problem is related to the coarse granularity at which changes are checked in and, consequently, seen by others. When trying to understand the evolution of the code, the delta between subsequent changes can be complex enough to prevent developers from inferring the design decision behind the changes in the code. Moreover, as indicated by previous studies [8, 3], large commits can also lead to merge conflicts, duplicated work, and conflicting design decisions .

In our previous work [10, 12] we presented Syde, an Eclipse plug-in that records fine-grained changes in multi-developers projects by continuously tracking code edits performed in the Integrated Development Environment (IDE).

In recent work we presented Replay [13, 11], an Eclipse plug-in that allows developers to explore the rich change repository created by Syde. Developers can search for fine-grained changes made by a set of people to a set of artifacts and watch them in the chronological order as originally performed in the IDE. This counts for a better user experience [21] than the aggregated form of commit-based Software Configuration Management (SCM) tools, such as CVS and Subversion.

In a previous version of this paper [11], we conducted a controlled experiment to assess whether Replay is at least as effective and efficient as the state of the practice at supporting developers with their questions related to software evolution [11]. The design of the experiment involved the selection, from previous catalogues [26, 2, 6, 15], of a set of common questions that developers ask. We converted them into a set of tasks to measure both the correctness of the task solutions and their completion time. In this extended version of the paper, we conducted additional runs of the controlled experiment, involving new participants, we expand our analysis of the experiments results, and we make our experiments replicable by sharing the necessary information.

The contributions of this article are:

1. *Replay*, a toolset to replay and exploit a fine-grained change software repository, thus aiding developers in answering their questions related to software evolution;
2. a report on the design and operation of a series of controlled experiments to compare the performance of Replay with the baseline tool (SVN client) in performing selected software evolution comprehension tasks;
3. an analysis of the results, which shows a statistically significant advantage of Replay over the baseline in time, and indicates advantages of Replay on correctness;
4. the complete experimental data to make our experiment replicable.

Structure of the article. In Section 2 we review Syde and its change model to subsequently present Replay. In Section 3 we describe the design and operation of our controlled experiment. In Section 4 we analyze the experiment results and discuss the

threats to validity. In Section 5 we present work related to the tool, and to the controlled experiment. In Section 6 we present the concluding remarks. Finally, in Appendix A we present the complete dataset that makes this experiment replicable.

2 Tool Support: Syde and Replay

2.1 Syde

Syde is a client-server application that records fine-grained information about the evolution of a system developed in a multi-developer setting [10, 12]. It extends Robbes' change-based software evolution (CBSE) model [24] into a multi-developer context by modeling the evolution of a system as a set containing sequences of changes, where each sequence is produced by one developer. A change takes a developer's copy of the system from one state to the next by means of semantic operations. These operations are captured by Syde's client, an Eclipse plug-in, triggered at every build action. Thus, the evolution of a system comprises the combination of the sequences of changes produced by each individual.

System Representation Syde models and captures changes of Java systems. It stores and analyzes constructs such as classes and methods, instead of files and lines. To this aim, a system is modeled as an abstract syntax tree (AST) containing nodes—which represent packages—classes, methods, and fields. In a multi-developer project, the current state of a system is different for each developer, as it depends on the changes each has performed after a checkout. The current state of a system is therefore represented by keeping track of one AST per developer.

Change Operations In CBSE, change operations represent the evolution of the system instead of file versions. A change operation is the representation of a change a developer performs in the workspace, *i.e.,* it is the transition of a system from one state to the next. Syde captures both atomic changes and composite change operations (*e.g.,* refactorings [5]). Atomic changes (*e.g.,* insertion, deletion and change of the property of a node) are the finest-grained operations on a system's AST, and contain all the necessary information to update the model. By applying a list of atomic changes in their chronological order, it is possible to generate all the states of a program's evolution.

System Architecture Syde is a client-server application, in which the server records the change operations, maintains the current state of a project and publishes information about current and past activities of the team. The client is a collection of plug-ins that enriches the Eclipse IDE to track changes and to show awareness information to developers.

2.2 Replay

Replay is one of the plug-ins that compose Syde's client. Its goal is to allow developers to explore the evolution of a system by chronologically replaying the changes collected

by Syde. Since atomic changes are too fine-grained to be shown individually, Replay groups them by timestamp, author and artifact (package or class), *i.e.,* all the changes that were performed by a developer in a class between two subsequent builds are grouped together based on the last build's timestamp. Within a group there cannot be more than one change to one artifact, thus we maintain the granularity of the changes.

Change Groups Each change group contains the following information:

- the set of changed artifacts, which can be packages, classes, methods, or fields;
- the type of change for each artifact, which can be insertion, deletion or change;
- the timestamp of the change, more precisely of the build in which the changes in this group were captured;
- the author of the changes,
- the SCM revision that was the baseline for the change.

Change Filters To help developers address different problems, Replay offers three orthogonal categories of filters applicable to the changes of a system under analysis:

- *Time-based.* They filter the changes based on the time period in which they were performed, specified as a combination of begin and end time.
- *Artifact-based.* They focus the replay on a subset of the system artifacts, i.e., classes or packages.
- *Author-based.* They focus the replay on the activity of a subset of the authors in the system. Such a subset can be a team of developers, or a single person.

Visualizing Changes Figure 1 presents the main components of the Replay plug-in:

- *The Replay View* (Point 1) lists the changes resulting from a search. It shows the entity from the group that is the upper-most node of the AST model of these changes. The change description refers to the entity shown. The other pieces of information provided are the timestamp, author and SCM revision for that group of changes. Selecting one change in the list determines the code to be displayed on one of the editors of the user's choice.
- *The Replay Editor* (Point 2) shows the source code of the change selected on the Replay View. It colors different types of changes with different colors. In the example from Figure 1, the orange text indicates that there was a change on the signature of the constructor in class `MainFrame`. Alternatively, the user can also switch to the Compare Editor (by clicking on Point 3) to view the changes, which shows the structural and textual comparison between the change selected on the Replay View and the prior change.

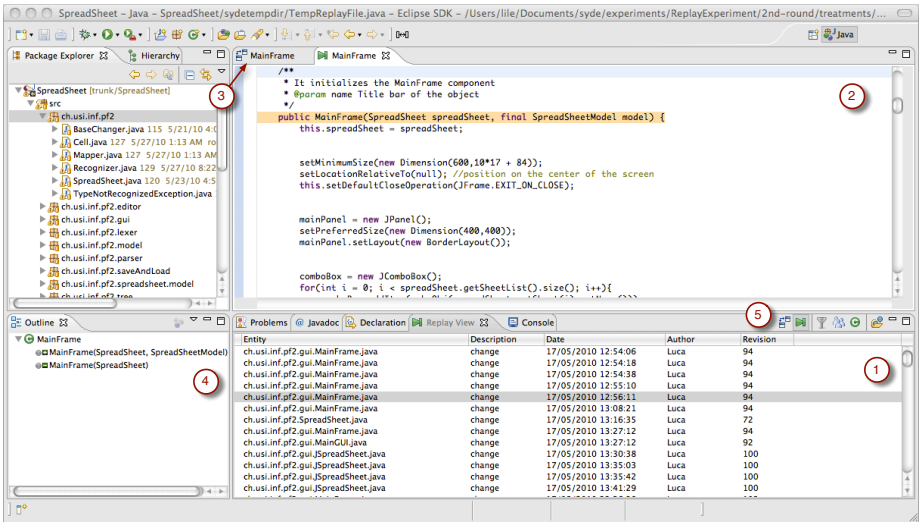


Fig. 1: The main components of the Replay plug-in

- *The Customized Outline View* (Point 4) is a complement to the information shown in the Replay Editor. It gives structural information about the highlighted changes. In the example in Figure 1, it indicates that a parameter was added in the signature of the constructor `MainFrame`.
- *The Toolbar* (Point 5) allows one to (1) choose the way in which the changes are displayed in the main editor (the first two icons), (2) filter changes based on criteria like author, artifact, or time (the next three icons), and (3) improve tool performance by caching the changes locally instead of accessing the server for every search (last icon).

To watch the changes replayed in chronological order, the user can navigate through the change list in the Replay View and observe the information shown on the Replay (or the Compare) Editor and in the Outline View. Watching a development session gives the user access to which classes were changed, which parts within the classes were changed, the order in which they were changed, by whom, *etc.*

3 Experimental Design

We want to quantitatively evaluate the effectiveness and efficiency of Replay on helping developers to answer questions they ask while developing software. The developer questions we focus on are related to the evolution of the system and can be answered by analyzing data from source code repositories.

3.1 Research Questions and Hypotheses

We raise the following research questions:

RQ1 Does the use of Replay reduce the **time** for answering software evolution questions compared to SCM-based tools?

RQ2 Does the use of Replay increase the **correctness** of the answers to software evolution questions compared to SCM-based tools?

RQ3 Does the user's **experience level** affect the potential benefits of using Replay in terms of correctness and time?

RQ4 Which **type** of questions can we identify that benefit most from the use of Replay?

The null and alternative hypotheses associated with the first two questions are formulated in Table 1.

Tab. 1: Null and alternative hypotheses

	Null Hypotheses		**Alternative Hypotheses**
$H1_0$	The tool does not impact the time required to answer software evolution questions.	$H1$	The tool impacts the time required to answer software evolution questions.
$H2_0$	The tool does not impact the correctness of the answers to software evolution questions.	$H2$	The tool impacts the correctness of the answers to software evolution questions.

To test the hypotheses $H1_0$ and $H2_0$ we define a series of tasks that have to be performed by the control and the experimental group. The control group (Eclipse+SVN) uses an Eclipse installation with default development tools and Subclipse[1] to answer the questions accessing the change history from SVN. The experimental group (Eclipse+Replay) uses the same Eclipse installation with default development tools and Replay to access Syde's change history. We maintain a between-subject design, meaning that each subject is part of either the control or the experimental group.

To answer RQ3 we analyze the data within blocks to check whether the experience level influences the participants' performance. For the last question we perform a separate analysis of correctness and completion time for each task.

3.2 Object System

The system we chose to be the object of this experiment is called *SpreadSheet*. Developed by a team of four BSc students, it is a simple spreadsheet application with support for basic mathematic formulae. Its development lasted for six weeks, and at the end, the project counted 13 packages, 77 classes, 286 methods, totaling 1,882 lines of Java code. The number of SVN commits was 137, while the number of recorded Syde changes was 11,661. The choice of this specific system is constrained by the need of having the change history, collected from both Syde and SVN, for the entire development cycle. Thus, it was neither possible to choose an open-source system, nor did we have a team

[1] Subclipse provides support for SVN in Eclipse `http://subclipse.tigris.org/`

working on a commercial system at our disposal. This choice implies some threats to the validity of the experiment, discussed in Section 4.8.

3.3 Task Design

We want to evaluate whether Replay outperforms a baseline (Subclipse) for assisting developers in answering comprehension questions related to a system's evolution.

We considered previous catalogues of questions [2, 6, 15, 26], selected those that can be answered by investigating the change history of the system, and created corresponding tasks. Table 2 provides a short description of each task together with its goal and rationale. Each task is an adapted version of a question from a previous catalogue [6].

Tab. 2: Tasks' description, goal and rationale

Id	Description	Goal	Rationale
1	Imagine that you are joining the project's team to replace a member. Find out what he was working on, so you can start from what he left unfinished. Identify the two classes he changed the most in the past days.	Becoming familiar with someone else's work	It is not uncommon that developers leave and join the team/company during the development of a software system. The goal of this task is to simulate when a newcomer has to take over the responsibility of a developer who just left the team.
2	You have just started to work on a set of classes, and want to find out whether someone else has recently changed them before you commit your changes. Identify the methods that someone else has also changed.	Becoming aware of team activity	Developers are not simply interested in knowing who is working on what, but they are rather interested in knowing who is working on parts of the system that can impact on their work (or that their work can impact on).
3	You have identified one of the main classes of the system but cannot quite understand it. Look for experts who can help you by searching who has recently changed it the most.	Finding experts at the class level of abstraction	Developers often find themselves trying to understand a part of the code that was written by someone else. The goal of this task is to simulate how a developer would find out who to ask for help on a class.
4	You are taking over the responsibility of a class and your first task is to refactor the code to improve its design and readability. You want to start with the most complicated feature, because it will need most of your effort. From the list below, identify the feature that provoked the largest number of changes.	Relating a feature to code changes	Developers are often confronted with the task of making a part of the source code more readable and maintainable, usually through refactorings. In order to do so, they need to understand the code, and identify the most problematic parts of it, the design flaws, etc. This task tackles the identification of parts of the code that need more attention.
5	You are given the description of a defect and instructions to reproduce it. Find out the origin of this defect, when and by whom it was introduced. Propose a fix.	Tracking back the introduction of a defect	Resolving defects is part of every developer's job. The goal of this task is to resolve a defect by tracking back when it was introduced and reverting the changes.
6	Before you joined the team the system underwent a major refactoring, which involved the deletion of a class and restructuring of other classes. Investigate why this refactoring took place.	Understanding the rationale behind past refactorings	Decisions taken during the development of a system are seldom documented. The goal of this task is to simulate a situation in which one needs to understand the rationale behind an undocumented refactoring.

In the handout distributed to the subjects, the descriptions of the tasks are integrated in the following hypothetical scenario: The subject is joining a team to replace a developer that left. The scenario makes the subjects feel as if they have become part of the team and are gradually learning the system while solving the experimental tasks.

3.4 Subjects

We conducted the experiment with 45 subjects: 18 MSc students, 25 PhD students, one postdoc, and one professor. The participant's average age was 27.84, comprising 14 different nationalities. The MSc students, the postdoc, and the professor have a software engineering background. The PhD students have also other backgrounds, such as information retrieval, human-computer interaction, and security. Participation was on a voluntary basis. None of the participants had previous experience with Replay.

3.5 Operation

The operation is composed of several experimental runs. Each run includes a training session of *ca.* 15 minutes and one experimental session. Each training session consists of a tutorial on the tool usage given by the experimenter, followed by a hands-on session, where the subjects perform some small tasks and can ask clarification questions. The experimental session is composed of six tasks, with time limit as follows: 10 minutes for each of the first four tasks, 20 minutes for task five, and unlimited time for task six.

The session is conducted by using the subject's laptop, and instructions are provided to configure it for the experiment. The control group had Eclipse and Subclipse installed; a local SVN server was provided. For the experimental group, the subjects were asked to install Eclipse and Replay.

There were 12 experimental runs in four locations: one run with seven participants at the University of Berne; one with six participants at the University of Zurich; four with two participants, two with one participant, one with six participants and one with 13 participants at the University of Lugano; one with two participants, and one with one participant at the University of British Columbia. Table 3 summarizes the 12 experimental runs.

3.6 Pilot Studies

To refine the experiment design and make Replay mature enough to guarantee an operation without technical impediments, we ran 5 pilot studies involving 19 people over the course of 4 months. Regarding Replay, we put most of the effort in improving its performance on retrieving the change history to be comparable to Subclipse.

Several experimental parameters were adjusted after each pilot, including the description and quantity of the tasks, the time limit of each task, the computer configuration, and the handout. One of the parameters that was carefully evaluated was the slowdown caused by the configurations of the tools, which led to the experimental group running Syde's server locally, and the control group having access to a local SVN repository.

Tab. 3: Summary of the experimental runs

Date	Location	Participants
10.12.2010	University of Lugano	1 PhD
13.12.2010	University of Lugano	4 MSc, 2 PhD
17.01.2011	University of Berne	1 MSc, 5 PhD, 1 Professor
13.01.2011	University of Lugano	1 PhD
13.01.2011	University of Lugano	1 PhD, 1 Post-doc
18.01.2011	University of Lugano	2 PhD
19.01.2011	University of Lugano	2 PhD
25.01.2011	University of Lugano	2 PhD
28.01.2011	University of Zurich	6 PhD
23.08.2011	University of British Columbia	1 PhD
24.08.2011	University of British Columbia	2 PhD
21.10.2011	University of Lugano	13 MSc

3.7 Data Collection

We collected four different types of data during the experiment:

1. *Personal information.* Before the experiment, we collected, through a screening questionnaire, information about the subject (*e.g.,* age, affiliation) and the subject's experience with Java, Eclipse and Subversion.

2. *Timing data.* To time the participants, we adopted two strategies. When the session involved up to two subjects, the experimenter timed them manually. When the session had more than two subjects, the experimenter used a timing web application to time each subject, and also to show them their remaining time. In both cases, the experimenter notified the subjects when they went overtime, and allowed them to write down their findings before going to the next task.

3. *Correctness data.* To convert the solutions into quantitative values, we established a grading system. Each task is worth 1 point, evenly distributed according to the number of correct answers that must be entered., *e.g.,* if there are 4 correct answers, each is worth 0.25, while each wrong answer counts as -0.25. The correct answers were determined by the experimenter, and double-checked by two other persons.

4. *Participant feedback.* The experiment ended with a debriefing questionnaire, where the subjects assessed the time pressure, the difficulty of the tasks and whether the tasks were realistic. The subjects were also given the opportunity to write down their opinions about the experiment and the tools.

4 Analysis and Interpretation

We performed a preliminary analysis on the oppinions of the subjects regarding the tasks, to check for exceptional conditions.

Difficulty We asked the subjects to indicate how much time pressure they felt during the experiment from 1 (no pressure) to 5 (too much pressure). The average time pressure reported is 2.75 (stdev. 1.02) for the control group and 2.55 (stdev. 1.05) for the experimental group, who felt slightly less time pressure. We sorted the tasks in increasing order of difficulty throughout the experiment, which is confirmed by the subjects' assessment in Figure 2.

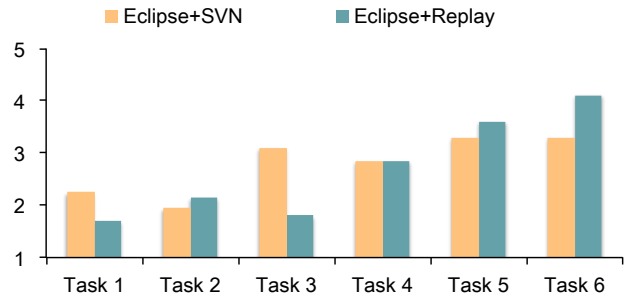


Fig. 2: Difficulty: 1 - trivial, 2 - simple, 3 - intermediate, 4 - difficult, 5 - impossible

Although there is a great difference on the perceived difficulty between control and experiment groups in tasks 3 and 6, they do not characterize a high discrepancy in terms of both completion time and correctness, thus we decided to maintain these tasks in the analysis. Since task 6 required a subjective answer in the form of a short essay, it is not included in the statistical test.

Realism As shown in Figure 3, the participants felt that the tasks reflect situations that happen in real development scenario.

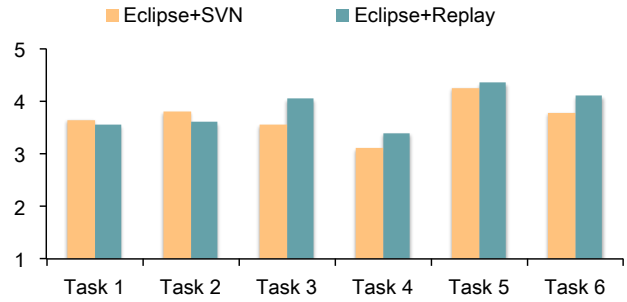


Fig. 3: Realism: 1-strongly disagree, 2-disagree, 3-undecided, 4-agree, 5-strongly agree

Task 4 received the lowest grading, especially from the control group. We believe this is due to the formulation of the task description rather than the task goal.

4.1 Subject Analysis

We followed the suggestions of Wohlin *et al.* [31] regarding the removal of outliers caused by exceptional conditions before performing our statistical test. We registered a number of outliers.

One subject from the control group was unable to finish the experiment in the allotted time due to lack of experience with Eclipse. One subject from the experimental group did not follow the instructions provided in the handout regarding the tools he was allowed to use, and used the tools reserved to the control group instead. One subject from the experimental group did not understand the concept of fine-grained changes provided by Replay. His answers clearly showed that he did not use the tool, but rather answered randomly, characterizing himself as an outlier both in terms of correctness (low grading) and time (low completion time). Finally, two subjects (one from each group) failed to register the completion time of at least one of the tasks.

We excluded these five cases from the statistical analysis, and were left with 40 subjects. We previously assigned treatments to subjects using randomization and blocking according to their experience level. We asked the subjects to indicate the number of years of experience they have in programming in Java, using Eclipse, and using SVN. The criterium used for the blocking was: A subject is considered advanced only if he has at least four years of experience with Java and Eclipse, and at least one year of experience with SVN. If one of these criteria is not met, the subject is classified as beginner. As a result of the random assignment and after the removal of the outliers, we obtained a fair distribution of subjects, as shown in Table 4.

Tab. 4: Subject distribution

	Eclipse+SVN	**Eclipse+Replay**	**Total**
Beginner	12	11	23
Advanced	8	9	17
Total	20	20	40

4.2 Interpretation of the Results

The design of our experiment is a between-subjects with balanced design, and one independent variable, *i.e.,* the tool. The choice of the hypothesis test depends on whether the sample distributions are normal and have equal variances. If it meets these two requirements, we can choose the parametric Student's t-test, otherwise, we should use the nonparametric Mann-Whitney U test.

We performed the Shapiro-Wilk test of normality, which only rejected the hypothesis that the experimental sample for correctness is normal (p-value $= 0.046 < 0.05$). For completion time, we also performed the Levene test and verified that the samples have equal variances. The descriptive statistics related to correctness and completion time are presented in Table 5.

The results of the statistical tests are presented in Table 6. Since the completion time is normally distributed with equal variances, we use the Student's t-test for its analysis. For correctness we must use the Mann-Whitney U test.

Tab. 5: Descriptive statistics of the experiment results

	Group	Mean	Diff.	Min.	Max.	Stdev.
Time (minutes)	Ecliple+SVN	47.71		36.20	55.55	5.98
	Ecliple+Replay	42.19	-11.56%	32.00	53.92	5.24
Correctness (points)	Ecliple+SVN	3.52		1.00	5.00	1.04
	Ecliple+Replay	4.11	+16.76%	2.33	5.00	0.81

Tab. 6: Results of the statistical tests

	Group	S-W		Student's t-test			MWU
		p-value	Levene	t	df	p-value	p-value
Time (minutes)	Ecliple+SVN	0.091					
	Ecliple+Replay	0.855	0.144	2.762	38	0.009	
Correctness (points)	Ecliple+SVN	0.065					
	Ecliple+Replay	0.046					0.062

4.3 Results on Completion Time

We first test the null hypothesis $H1_0$, which states that the use of the tool Replay does not impact the time required to complete the assigned tasks.

Table 5 shows that the experimental (Eclipse+Replay) group took on average 11.56% less time to complete the tasks than the control (Eclipse+SVN) group, and that this result is statistically significant at the 99% confidence interval (p-value $= 0.009 < 0.01$ for the t-test). With these results, we can reject the null hypothesis $H1_0$ in favor of the alternative hypothesis $H1$, and positively answer RQ1.

Figure 4 shows a box plot[2] of the total time (in minutes) spent by the subjects on the first five tasks. As we can see, the 50th percentile of the experimental group is roughly at the same level of the 25th percentile of the control group. This means that almost 50% of the subjects from the experimental group completed the tasks before or at about the same time as 75% of the subjects from the control group. **Mir** ▶*Isn't this the other way around?*◀

The variability (or range) of completion time is slightly higher in the experimental group than in the control group. One factor that might have influenced this result is that Replay was unknown to everyone, while most of the subjects had some experience with SVN. In addition, some subjects spent a long time getting used to Replay while doing the warm up task, and asked for help when they were struggling with the tool, while others quickly completed the warm up task, and seldom asked for help. Therefore, previous experiences of the subjects of the experimental group with using other tools (including SVN itself) and their dedication on understanding Replay before starting the tasks might have resulted in a higher variability in completion time.

The fine granularity and the large amount of changes that the experimental group had to look through were common complaints, which might have influenced the completion time of the group. However, these "drawbacks" of the Replay tool did not prevent the

[2] The right end of the box represents the 75th percentile, the left end of the box the 25th percentile, and the line in the middle the 50th percentile (median).

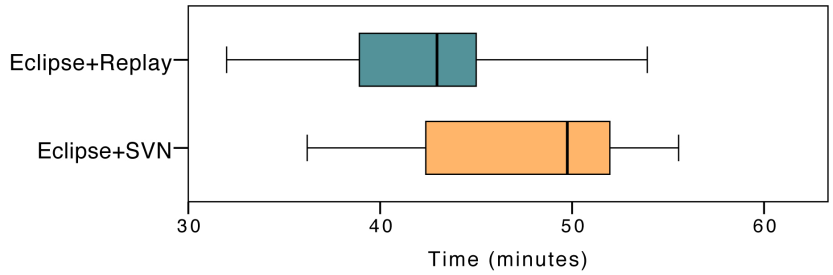


Fig. 4: Results on completion time.

experimental group to outperform the control group in terms of completion time.

4.4 Results on Correctness

Table 5 shows that the experimental group obtained, on average, a score 16.76% higher than the control group. However, this result is not statistically significant at the 95% confidence interval (p-value $= 0.062 > 0.05$ for MWU test), but it is at the 90%. We are unable to fully reject the null hypothesis $H2_0$, and partially answer RQ2.

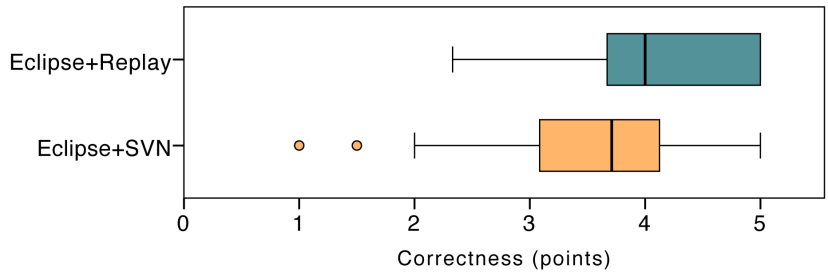


Fig. 5: Results on correctness.

Even though the results are not statistically significant at the 95% confidence interval, Figure 5 shows evidence that the experimental group had a superior performance than the control group. The 25th percentile of the experimental group is at the same level of the 50th percentile of the control group, *i.e.,* 75% of the subjects from the experimental group obtained higher (or equivalent) score than 50% of the control group subjects.

Figure 5 shows that there were two outliers in the control group, who scored 1 and 1.5 out of 5. These subjects (C16 and C19) are classified as beginners, as both have little experience with the tools used (1 to 3 years of experience with Java and Eclipse, and less than 1 year of experience with SVN). They rated themselves as beginners with using SVN, which is an evidence that they might not be familiar with the tool, although they could indicate they had no familiarity with it (when a developer indicated no previous

experience with SVN, he was automatically assigned to the experimental group).

At a closer inspection we found no sign that their answers were randomly selected, because some of the mistakes were also common to other subjects. For instance, for task 4 both answered that the changes were most related to "handling mouse events", which was the number 1 mistake of both groups. In addition, when they went overtime, they left the answer sheet blank (*e.g.,* task 5 for C16 and task 1 for C19), which supports our analysis that they did not answer randomly, but would rather need more time to answer some questions because of their low experience with the tool. Therefore, these subjects were not classified as outliers and were kept in the analysis.

Non-parametric tests (*e.g.,* MWU) are more conservative than parametric tests (*e.g.,* t-test), meaning that non-parametric tests need more samples or greater difference in the values to yield statistically significant results. We argue that this is the main reason for the MWU test to have retained the null hypothesis at 0.05 significance level. One evidence of the tendency to reach statistically significant results with more samples is that the results reported here—with 40 samples—improved in comparison with previous results, when there were 26 samples [11] (p-value decreased from 0.072 to 0.062).

4.5 Influence of the Experience Level

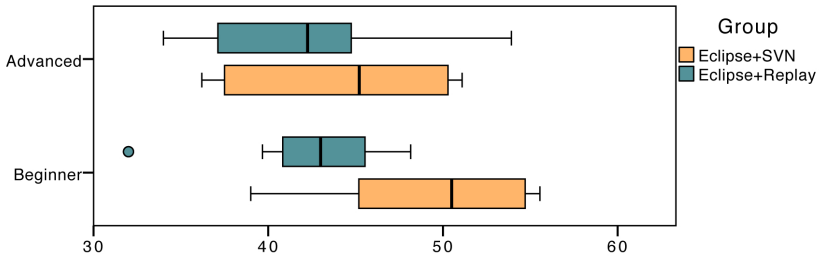
We compared the correctness and completion time across the two levels of experience, i.e., beginner and advanced. Figure 6 shows that the experimental group outperformed the control group in both correctness and completion time, regardless of the experience level. Even though the number of subjects per experience level is too low to yield statistically significant results, we draw a couple of observations based on the box plots.

The variability of the experimental group was lower than the one of the control group for the beginners, while the inverse can be observed for the experts in terms of completion time. Our assumption is that in the case of beginners, since both Replay and Subclipse are new to them, the learning curve of Subclipse is steeper than the one of Replay. For those unfamiliar with Subclipse, we gave a tutorial on its usage and allowed the subjects to get used to it before starting to perform the tasks. An interesting feedback we collected from some experts is that they felt they were so used to look at the changes the "SVN way" that it was not easy to adapt to Replay, which hindered their performance. The higher variability in completion time observed in the advanced-experimental group can also be explained by their efforts to adapt to Replay.

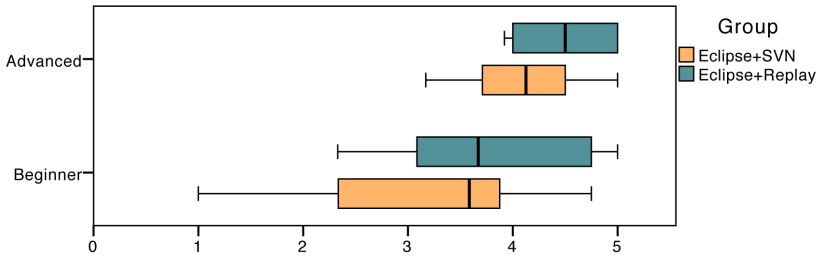
One outlier appears on the beginner-experimental group for completion time. Moreover, this subject answered all questions correctly. Thus, regardless of his low experience, he was efficient and effective, which might characterize him as a fast learner.

In terms of correctness, both beginners and advanced from the experimental group had lower variability than their respective from the control group. A factor we attribute to this result is the coarse granularity of the information contained in the SVN repository, which can be the subject of multiple interpretations.

We can answer RQ3 by stating that the users' experience level does not affect the potential benefits of using Replay in terms of correctness. However, the results suggest that the users' experience might influence their efficiency.



(a) Time (minutes)



(b) Correctness (points)

Fig. 6: Beginner versus advanced

4.6 Individual Task Analysis

To identify which type of tasks can benefit most from the use of Replay (RQ4), we examine the performance of the two tools per task. Figure 7 shows the average correctness and completion time for each task.

Task 1 – Becoming familiar with someone else's work The subjects are asked to familiarize with recent changes made by one developer, and to identify the two classes this developer worked on the most. The experimental group achieved an excellent performance, while the control group took more time and had lower grading. Parnin and DeLine have observed that when a developer resumes one of his interrupted tasks, he prefers to see past changes chronologically than in aggregated form [21]. Our findings complement theirs by showing a better performance of those seeing the changes made by others chronologically.

Task 2 – Becoming aware of team activity The goal of this task is to identify which methods were recently changed and by whom. Although we have developed a plug-in that directly targets awareness [18], Replay can also be used for this activity. The results show that Replay is slightly more effective and as efficient as the baseline.

Task 3 – Finding experts at the level of abstraction of classes In this task, the subjects are asked to identify experts for a class based on the recent changes it

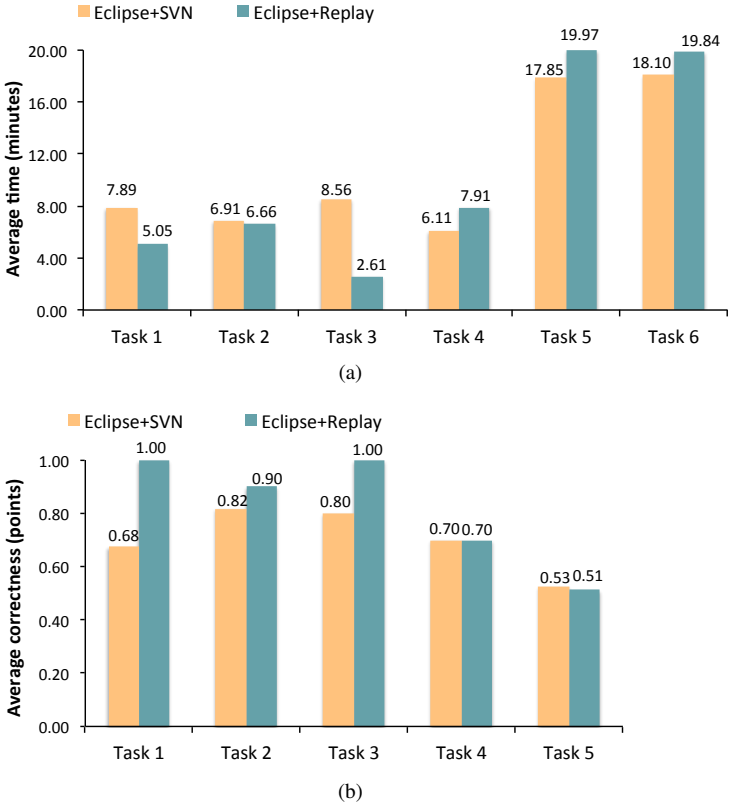


Fig. 7: Average completion time and correctness per task

underwent. The results are very similar to the ones from task 1, and show that Replay outperforms the baseline for this task.

Task 4 – Relating a feature to code changes This task involves identifying the different features inside a class and identifying the one that has changed the most. The experimental group took, on average, more time than the control group to complete the task, and had equal effectiveness. A possible explanation for this result is that number of fine-grained changes the subjects from the experimental group had to look at was higher than the number of SVN commits the control group had to inspect.

Task 5 – Tracking back the introduction of a defect This task was added to the experiment after collecting suggestions on the pilot study indicating that Replay could be useful for identifying the change that caused a defect. The control group performed better than the experimental in terms of completion time, and the two groups had similar performance in relation to correctness. The results and the feedback collected from the

experimental group indicate that one of the reasons for them to taking longer than the control groups is that there were too many fine-grained changes to inspect for this task, being counter-productive.

Task 6 – Understanding the rationale behind past refactorings The goal is to understand the design decisions behind a major refactoring performed in the past. Since this is a task that requires a descriptive answer, only the quantitative scores of completion time are available. On average, the control group was slightly faster to complete the task than the experimental group. We have given a qualitative grading to the answers, shown in Figure 8. The essays confirm that the control group was able to better understand the reasons behind the refactoring, with the majority of the very satisfactory answers being from them. The answer below exemplifies such an answer given by a control group subject:

"The functionality of Sheet was moved to SpreadSheetModel.java. It contains now the cell-grid. Before it only stored the cell contents as String rather than the cell objects. For that all other classes that referred to Sheet before had to be changed: e.g., from "Sheet sheet" to "SpreadSheetModel sheet" in SpreadSheetWriter.java." (C13).

It shows that the subject understood beyond the essence of the refactoring, which was the replacement of the functionality of Sheet in SpreadSheetModel. The subject also understood the design decision behind the refactoring, and how this change affected the system's related classes. Similarly, the answer below shows a very satisfactory answer from a subject of the experimental group:

"Before refactoring, the classes Sheet and SpreadsheetModel were separated, and one could convert a Sheet to a SpreadsheetModel by calling a function. The refactoring eliminated the need for these two separated classes, by merging the functionalities of both only in one of them (SpreadsheetModel). I guess it was done to keep consistency of the state of these entities, which were treated separately, but their state depended on each other." (E14).

The answers classified as satisfactory and somewhat satisfactory are those in which the subjects identified the classes directly involved in the refactoring (`Sheet` and `SpreadSheetModel`), but did not understand why the refactoring happened. The major differences between them are that satisfactory answers also identified classes that are indirectly involved in the refactoring and somewhat satisfactory answers express doubts about the refactoring. An example of a satisfactory and a somewhat satisfactory answer are shown below:

"Sheet and SpreadSheetModel were essentially duplicates. So the former was deleted and replaced by the later. The code from Sheet was merged into SpreadSheet-Model. MainFrame changed a lot, and as a result, the model classes had to be modified. That's how the duplication received attention." (E7 - satisfactory).

"In the Sheet class, the convert to SpreadSheetModel has been changed multiple

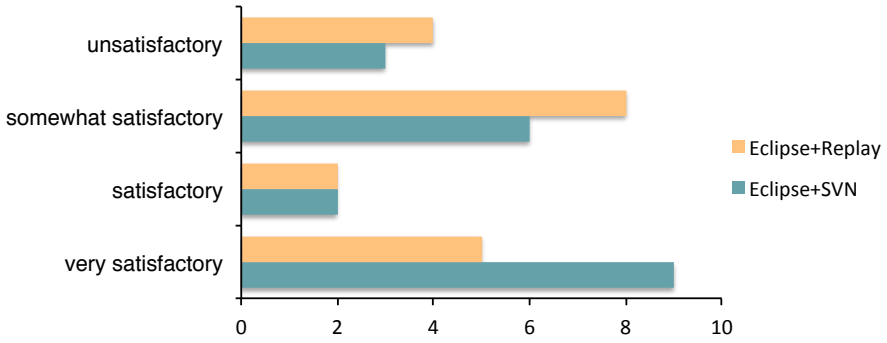


Fig. 8: Grading of the answers to task 6

times. SpreadSheetModel and SpreadSheet class have been changed. I think the reason was because of the way sheets were kept in an array and converting them to a SpreadSheet. I did not understand very well." (E12 - somewhat satisfactory).

Finally, the unsatisfactory answers are those that fail to identify the classes directly involved in the refactoring, as the example below:

"The refactoring seems to include the solution of more comments on the classes (i.e., the authors). Sheet has been replaced by SpreadSheetSelectionModel." (C15).

As previously stated, the results from Figure 8 and the content of the essays suggest that subjects from the experimental group had more difficulty to understand the refactoring than those of the control group. Another evidence comes from the difficulty level of the task collected on the debriefing and shown in Figure 2. It indicates that the experimental group struggled to understand the refactoring (the average difficulty was higher than 4), while the average difficulty indicated by the control group was 3 (intermediate).

Similar to task 5, the experimental group complained that the information provided by Replay was too fine-grained to allow them to see the "big picture" of the refactoring.

Summary For tasks 1 and 3, which needed fine-grained information about recent changes, Replay was more efficient and effective than the baseline. For task 2, when the subjects had to relate recent changes to authors, Replay was more effective, and as efficient as the baseline. For task 4, in which measuring the amount of work done in different parts of the code was important, Replay proved to be less efficient but just as effective. At tasks that required analyzing information over a long time span (tasks 5 and 6), the fine-granularity of Replay prevented the subjects from performing better than the baseline. To overcome this issue, as part of our future work, we plan to add a feature in Replay that offers the possibility to aggregate fine-grained changes using a

customizable aggregation factor.

4.7 Subjects' Feedback

In the debriefing questionnaire we asked the subjects from the experimental group to leave comments and suggestions on the Replay plug-in. In this section we summarize their feedback, which can be classified into three groups: (i) defects or flaws on the plug-in; (ii) suggestions strongly related to the tasks; (iii) suggestions to improve the tool.

Flaws and defects In general, the defects or flaws detected by the subjects impaired the user experience, but did not impede them to perform the tasks. One of the flaws that disturbed the subjects was the fact that sometimes, when browsing through the changed in the view, the tool froze. When this happened, the tool was actually requesting data to show on the view, however the response time was too long for the user's perception. We should have added a progress bar to indicate to the user that the process will take a few seconds to respond. A second flaw was that the tool did not allow subjects to open multiple editors. This was a design decision taken by the authors to allow the simulation of a video replay feature by showing the sequences of changes always in the same editor. However, the usage of the plug-in showed that opening multiple editors can be useful in situations that require the comparison of changes that are not subsequent.

Defects detected by the subjects included: when switching between editors (Replay and Compare), the tool failed to focus on the newly selected editor; and some descriptions of the changes did not correspond to the actual change. The latter is actually a defect of the tool responsible for capturing the changes, which is only shown when we replay them. The first defect was detected during the first experimental runs, and was fixed for the the last runs.

Suggestions strongly related to the tasks Some suggestions were strongly tied to the tasks and if addressed they would have eased the path to the solution for the subjects. However, they would bring little benefit to the tool. For instance, one subject suggested saving snapshots of the view instead of having to request them again. Instead, we could keep a history of recent requests to allow for quicker selection of identical requests. Another example is one suggestion to make the Replay editor editable to allow for in-place code change (and specifically bug fixing for task 5). However, the idea of the Replay editor is to let users watch the history of changes, but not to change this history. This is the reason for the Replay editor to be non-editable.

Suggestions to improve the tool The subjects provided a rich list of suggestions to improve the tool and make it more usable. We will comment on them separating the suggestions per component.

In the Replay view, the most common suggestion was to provide a search bar to facilitate the search for a specific change. To be more useful, the search should include the content of the changes, so users can, for instance, search for a specific method name. Another request was to add a navigation mode that lets the user navigate through the

changes of a specific developer while still showing the changes of the other developers in the list. Two important requests point to opposite directions. Some subjects suggested to add a feature (*e.g.,* a time slider control) to allow for grouping of changes, because in some instances the changes were too fine-grained. The second request was to show even finer-grained changes, to the level of method, to facilitate the identification of what changed between subsequent versions of a class.

For the filters, the most common request was to memorize the previous dates selected by the user and add a reset button to reset the date at the user's will. In addition, the subjects requested to have a search bar to search for specific classes and users quicker. Users also suggested to keep a history of recent searches to facilitate the loading of similar requests.

In the Replay Editor, the only suggestion was to add markers on the ruler to indicate in which lines are the changes. This would facilitate navigation when multiple changes are present.

4.8 Threats to Validity

In this section we discuss the threats to internal and external validity. Threats to internal validity refer to influences that can affect the independent variable with respect to causality, without the researcher's knowledge [31]. Threats to external validity are conditions that limit our ability to generalize the results of our experiment [31].

4.8.1 Internal Validity

Subjects To reduce the threat that the subjects may not have been competent enough, we ensured that they had sufficient skills on the tools used during the experiment. To lessen the threat that the expertise of the subjects may not have been fairly distributed, we used randomization and blocking to assign treatments to subjects.

Tasks The tasks were designed by the authors of this paper, and thus may have been biased toward Replay. To mitigate this threat, we have based the tasks on valid questions that developers ask, which were reported in previous catalogues. The tasks might have been too difficult and the allotted time per task may have been insufficient. To alleviate these threats, we conducted several pilot runs to fine-tune them. Furthermore, the task that was classified as too difficult by the experimental group (task 6) was not designed to be included on the statistical analysis.

Experimental runs There were several runs and the differences among them may have influenced the results. However, the several pilot runs with different number of participants allowed us to have a stable and reliable experimental setup.

Training We provided a training session on Replay to all subjects of the experimental group, while the subjects from the control group were assumed to have familiarity with the baseline tools. To mitigate the fact that lack of proper training might have influenced

the results, we also provided a training session on Subclipse when a subject from the control group was unfamiliar with the tool.

4.8.2 External Validity

Subjects The fact that the subjects of the experiment were from academia may have limited our ability to generalize the results to the industrial environment. It is difficult to recruit practitioners who are willing to dedicate 2 hours of their time to do an experiment. To mitigate the lack of practitioners, we assume a relatively high average expertise level of the 40 selected participants. This assumption is sustained by the subjective assessment of the expertise provided by the subjects prior to the experiment. They were asked to rank their perceived knowledge according to the scale: 1–none, 2–beginner, 3–knowledgeable, 4–advanced, and 5–expert. The results—Java (avg. 3.65, stdev. 1.05), Eclipse (avg. 3.45, stdev. 0.90), SVN (avg. 2.90, stdev. 1.19)—indicate an average of knowledgeable subjects.

Tasks Our choice of tasks may not reflect real questions related to software evolution. This threat is neutralized by our reliance on existing catalogues [26, 2, 6, 15], which were mainly constructed through surveys and interviews with practitioners.

Object system The representativeness of our object system is an important threat, since it is a small system that was developed by undergraduate students, and may not reflect the complexity of large-scale industrial systems. The use of more than one object system may have yielded different results. However, the choice of the object system was constrained by the need of having the change history from both Syde and SVN.

5 Related Work

Approaches Related to Replay To our knowledge, Replay is the first tool to support replaying development sessions in a collaborative environment. However, other tools support programmers with their quests. Fritz and Murphy propose a prototype that combines different information fragments (source code, work items, change sets, teams, comments, wiki) to support 78 questions software developers ask about a development project [6]. The tool Ferret combines four different sources of information (static, dynamic, evolutionary, and Eclipse PDE) to build a knowledge base for answering conceptual queries [2]. James is a knowledge base, composed of IDE interactions and micro-blogging, to support developers with their quests [9].

Looking at our work in the broader context of software evolution, there are various lines of research that relate to ours. In the software evolution analysis context, several approaches make use of the changes performed to a system over its lifetime to support its comprehension: Lanza, Gîrba *et al.*, and Lungu *et al.* summarize and visualize respectively the evolution of classes [16], the evolution of class hierarchies [7], and the evolution of inter-module relationships [20]. In these works the history is not replayed, but summarized; and the order in which the changes are performed is lost. We specifically focus on replaying the change events in the order in which they happened.

A few approaches focus on replaying the changes that happened in a system. Wettel and Lanza visualize the evolution of the entire system by allowing the user to travel in time and observe the changes of the system as they are represented in a 3D city metaphor [29]. Hindle *et al.* present an animation of the evolution of the architecture of a system [14] in the Yarn tool. The animation presents the evolution of the relationships between the modules of the system. These approaches allow the animation of the changes, but present the changes at a high level of abstraction, from which the code is not accessible.

One major difference between our work and the ones aforementioned is the level of detail of the data. In most of the approaches the data is extracted from commit-based SCM systems, implying that changes between versions can be arbitrarily complex. An approach that uses fine-grained change information was proposed by Robbes [23]. Although he collects fine-grained information from software systems, Robbes does not use it to support the replaying of the changes, but he exploits it for other purposes, *e.g.,* to detect and characterize development sessions [25]. Dig, instead, uses a change-centric approach to record sequences of refactorings, and to replay them on other library-based applications [4].

Empirical Studies There are relatively few empirical studies by means of controlled experiments in software engineering. Further, there is no controlled experiment that directly relates to ours: answering developers' questions related to software evolution. However, there are a number of controlled experiments related to software evolution and program comprehension.

Quante evaluates, through a controlled experiment, the support provided by dynamic object graphs on answering a set of program comprehension tasks [22]. Cornelissen *et al.* performed a controlled experiment to evaluate the use of Extravis, an execution trace visualization tool, to answer program comprehension tasks [1]. Wettel *et al.* assess the use of CodeCity to perform program comprehension and quality assessment tasks [30].

The major difference between these controlled experiments and ours is that they evaluate tools that visualize data other than source code (dynamic graphs, execution traces, system models) to support program comprehension. We evaluate a tool that allows a developer to investigate the history of the system by looking directly at its source code.

6 Conclusion

We presented Replay, a tool that allows developers to explore the evolution history of a system by chronologically replaying the fine-grained changes collected by Syde. We argue that Replay can be useful to help developers in finding answers to questions they raise during development and maintenance that are related to the evolution of a system.

We conducted a controlled experiment to evaluate whether Replay is at least as effective and efficient as the state of the practice at supporting developers with their questions related to software evolution. The results indicate that Replay leads to an improvement in both correctness (16.76%) and completion time (11.56%), with the latter being statistically significant at 99% confidence interval. As an indication of a superior performance of the experimental group in terms of correctness, 75% of

this group performed better than or equal to 50% of the control group. In terms of completion time, 50% of the experimental group was faster than (or equivalent to) 25% of the control group. Mir ▶*50% faster than 25%??*◀ These results show that there are benefits in using Replay over the state of the practice tools for most of the tasks included in this empirical evaluation.

The per-task analysis of the results provided a number of insights on the type of tasks our approach supports best. For tasks that needed fine-grained change information, or in which the the inspection of recent changes was important, Replay outperformed the baseline. However, when the tasks required a high-level overview of the changes, Replay did not perform better than the baseline.

As future work, we plan to incorporate the suggestions given by the subjects to improve the usability of the tool. Some planned enhancements are to: offer the possibility to aggregate the changes when a general overview is needed; implement search bars in the Replay view and in the filters; provide a history of recent requests; and add a navigation mode to the Replay view that allows restricting the navigation to changes of a single developer.

Acknowledgment

We thank Alberto Bacchelli and Richard Wettel for helping us with the design; the subjects and the participants of the pilot study; Serge Demeyer, Harald Gall, Oscar Nierstrasz, Arie van Deursen, Anja Guzzi, Quinten Soetens, and Michael Würsch for helping us with local organizations. Hattori is supported by the Swiss Science foundation through the project "GSync" (SNF Project No. 129496).

References

[1] B. Cornelissen, A. Zaidman, and A. van Deursen. A controlled experiment for program comprehension through trace visualization. *IEEE Trans. on Software Engineering*, 99, 2010.

[2] B. de Alwis and G. C. Murphy. Answering conceptual queries with ferret. In *Proceedings of ICSE 2008 (30th Intl. Conf. on Software Engineering)*, pages 21–30. ACM Press, 2008.

[3] C. R. B. de Souza, D. Redmiles, and P. Dourish. Breaking the code, moving between private and public work in collaborative software development. In *Proceedings of GROUP 2003 (Intl. ACM SIGGROUP Conf. on Supporting Group Work)*, pages 105–114. ACM Press, 2003.

[4] D. Dig. *Automated Upgrading of Component-based Applications*. PhD thesis, University of Illinois at Urbana-Champaign, 2007.

[5] M. Fowler. *Refactoring - Improving the Design of Existing Code*. Addison-Wesley, 1999.

[6] T. Fritz and G. C. Murphy. Using information fragments to answer the questions developers ask. In *Proceedings of ICSE 2010 (32nd ACM/IEEE Intl. Conf. on Software Engineering)*, pages 175–184. IEE Computer Society, 2010.

[7] T. Gîrba, M. Lanza, and S. Ducasse. Characterizing the evolution of class hierarchies. In *Proceedings of CSMR 2005 (9th European Conf. on Software Maintenance and Reengineering)*, pages 2–11. IEEE CS Press, 2005.

[8] R. Grinter. Supporting articulation work using software configuration management systems. *Computer Supported Cooperative Work*, 5(4):447–465, 1996.

[9] A. Guzzi, M. Pinzger, and A. van Deursen. Combining micro-blogging and ide interactions to support developers in their quests. In *Proceedings of ICSM2010 (IEEE Intl. Conf. on Software Maintenance)*, pages 1 –5, 2010.

[10] L. Hattori. Enhancing collaboration of multi-developer projects with synchronous changes. In *Proceedings of ICSE 2010 (32nd ACM/IEEE Intl. Conf. on Software Engineering), Doctoral Symposium*, pages 377–380. IEEE CS Press, 2010.

[11] L. Hattori, M. D'Ambros, M. Lanza, and M. Lungu. Software evolution comprehension: Replay to the rescue. In *Proceedings of ICPC 2011 (19th IEEE International Conference on Program Comprehension))*, pages 161–170, 2011.

[12] L. Hattori and M. Lanza. Syde: A tool for collaborative software development. In *Proceedings of ICSE 2010 (32nd ACM/IEEE Intl. Conf. on Software Engineering)*, pages 235–238, 2010.

[13] L. Hattori, M. Lungu, and M. Lanza. Replaying past changes on multi-developer projects. In *Proceedings of IWPSE-EVOL 2010 (Joint 11th Intl. Workshop on Principles of Software Evolution and 5th ERCIM Workshop on Software Evolution)*, pages 13–22, 2010.

[14] A. Hindle, Z. M. Jiang, W. Koleilat, M. W. Godfrey, and R. C. Holt. Yarn: Animating software evolution. In *Proceedings of VISSOFT 2007 (4th Intl. Workshop on Visualizing Software for Understanding and Analysis)*, pages 129–136. IEEE CS Press, 2007.

[15] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proceedings of ICSE 2007 (29th ACM/IEEE Intl. Conf. on Software Engineering)*, pages 344–353. IEEE Computer Society, 2007.

[16] M. Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *Proceedings of IWPSE 2001 (4th Intl. Workshop on Principles of Software Evolution)*, pages 37–42. ACM Press, 2001.

[17] M. Lanza, S. Ducasse, H. Gall, and M. Pinzger. Codecrawler — an information visualization tool for program comprehension. In *Proceedings of ICSE 2005 (27th IEEE Intl. Conf. on Software Engineering)*, pages 672–673. ACM Press, 2005.

[18] M. Lanza, L. Hattori, and A. Guzzi. Supporting collaboration awareness with real-time visualization of development activity. In *Proceedings of CSMR 2010 (14th IEEE European Conf. on Software Maintenance and Reengineering)*, pages 207–216. IEEE CS Press, 2010.

[19] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of ICSE 2006 (28th ACM Intl. Conf. on Software Engineering)*, pages 492–501. ACM, 2006.

[20] M. Lungu and M. Lanza. Exploring inter-module relationships in evolving software systems. In *Proceedings of CSMR 2007 (11th IEEE European Conf. on Software Maintenance and Reengineering)*, pages 91–100. IEEE CS Press, 2007.

[21] C. Parnin and R. DeLine. Evaluating cues for resuming interrupted programming tasks. In *Proceedings of CHI 2010 (28th Intl. Conf. on Human Factors in Computing Systems)*, pages 93–102. ACM Press, 2010.

[22] J. Quante. Do dynamic object process graphs support program understanding? - a controlled experiment. In *Proceedings of ICPC 2008 (16th Intl. Conf. on Program Comprehension)*, pages 73–82. IEEE CS Press, 2008.

[23] R. Robbes. *Of Change and Software*. PhD thesis, University of Lugano, Switzerland, Dec. 2008.

[24] R. Robbes and M. Lanza. A change-based approach to software evolution. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 166:93–109, Jan. 2007.

[25] R. Robbes and M. Lanza. Characterizing and understanding development sessions. In *Proceedings of ICPC 2007 (15th IEEE Intl. Conf. on Program Comprehension)*, pages 155–164. IEEE CS Press, 2007.

[26] J. Sillito, G. C. Murphy, and K. D. Volder. Questions programmers ask during software evolution tasks. In *Proceedings of FSE-14 (14th Intl. Symp. on Foundations of Software Engineering)*, pages 23–34. ACM Press, 2006.

[27] M.-A. D. Storey. Theories, methods and tools in program comprehension: past, present and future. In *Proceedings of IWPC 2005 (13th Intl. Workshop on Program Comprehension)*, pages 181 – 191, May 2005.

[28] A. von Mayrhauser and A. M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28:44–55, 1995.

[29] R. Wettel and M. Lanza. Visual exploration of large-scale system evolution. In *Proceedings of WCRE 2008 (15th IEEE Working Conf. on Reverse Engineering)*, pages 219–228. IEEE CS Press, 2008.

[30] R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: A controlled experiment. In *Proceedings of ICSE 2011 (33rd Intl. Conf. on Software Engineeering)*, page to be published, 2011.

[31] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, 2000.

A Experimental Data

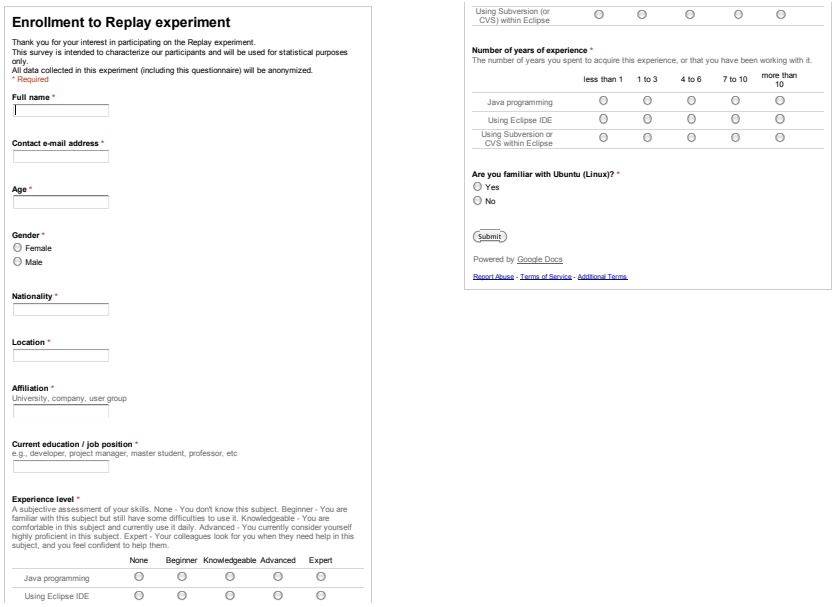
In this appendix we present all the details about our experiment, complementary to the ones presented in Section 4, which make our experiment repeatable: questionnaires, oracle sets, and the entire experimental data set collected from our subjects.

A.1 Object System

The Spreadsheet project used in this experiment is an open-source project under the GNU GPL v3 license. Its source code is available at `http://code.google.com/p/spread-ur-ca-gh-el/`.

A.2 Screening Questionnaire

Using Google Docs[3], we designed an online questionnaire that served both to provide an easily accessible platform for the volunteers to enroll and to allow capturing the personal information that we used to assign the subjects to treatments (See Figure 9).



Enrollment to Replay experiment

Thank you for your interest in participating on the Replay experiment.
This survey is intended to characterize our participants and will be used for statistical purposes only.
All data collected in this experiment (including this questionnaire) will be anonymized.
* Required

Full name *

Contact e-mail address *

Age *

Gender *
Female
Male

Nationality *

Location *

Affiliation *
University, company, user group

Current education / job position *
e.g., developer, project manager, master student, professor, etc

Experience level *
A subjective assessment of your skills. None - You don't know this subject. Beginner - You are familiar with this subject but still have some difficulties to use it. Knowledgeable - You are comfortable in this subject and currently use it daily. Advanced - You currently consider yourself highly proficient in this subject. Expert - Your colleagues look for you when they need help in this subject, and you feel confident to help them.

	None	Beginner	Knowledgeable	Advanced	Expert
Java programming					
Using Eclipse IDE					
Using Subversion (or CVS) within Eclipse					

Number of years of experience *
The number of years you spent to acquire this experience, or that you have been working with it.

	less than 1	1 to 3	4 to 6	7 to 10	more than 10
Java programming					
Using Eclipse IDE					
Using Subversion or CVS within Eclipse					

Are you familiar with Ubuntu (Linux)? *
Yes
No

Submit

Powered by Google Docs

Report Abuse - Terms of Service - Additional Terms

Fig. 9: The screening questionnaire used to co collect personal and experience information of the participants

[3] See `http://docs.google.com`.

A.3 Experimental Questionnaire

The content of the questionnaires, with the variations due to the different treatment combinations, is presented in the following. Their actual form and presentation is exemplified in Figure 10 and Figure 11, which show the questionnaire for the treatment T2.

Together with the description of the tasks, we also provide an oracle with the solution, specifying the grading per task.

A.3.1 Introduction

The aim of this experiment is to compare tool efficiency in supporting software practitioners understanding the change history of a software system.

You will use Eclipse with the <toolset> to analyze Spreadsheet, a spreadsheet application written in Java by undergraduate students at the University of Lugano.

You are going to Perform 6 tasks, with limited time to solve 5 of them. You have 10 minutes to solve each of the first 4 tasks, and 20 minutes to solve task 5.

We kindly ask you:

- to write down your answers in a legible way;
- not to consult any other participant during the experiment;
- to perform the tasks in the specified order;
- to write down the current time before starting to work on a new task (after reading it) and once after completing all the tasks;
- to announce to the experimenter that you are starting to work on a new task (after reading it), in order to reset your allocated timer;
- not to return to earlier tasks because it affects the timing;
- to fill in the required information for each task. In the case of multiple choices check the most appropriate answer and provide additional information, if requested.

In the following, there is one warm-up task for you to get used to the tool and the experiment.

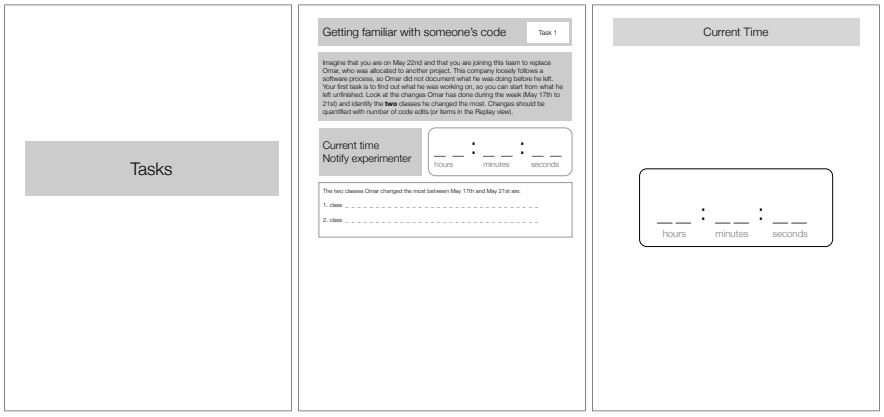
The experiment is concluded with a short debriefing questionnaire.

Thank you for participating in this experiment!
Lile Hattori, Michele Lanza, Mircea Lungu and Marco DAmbros

A.3.2 Tasks

Task 1: Getting Familiar with someone's code

(a) Introduction (b) Warm-up (c) Time - logged after each task



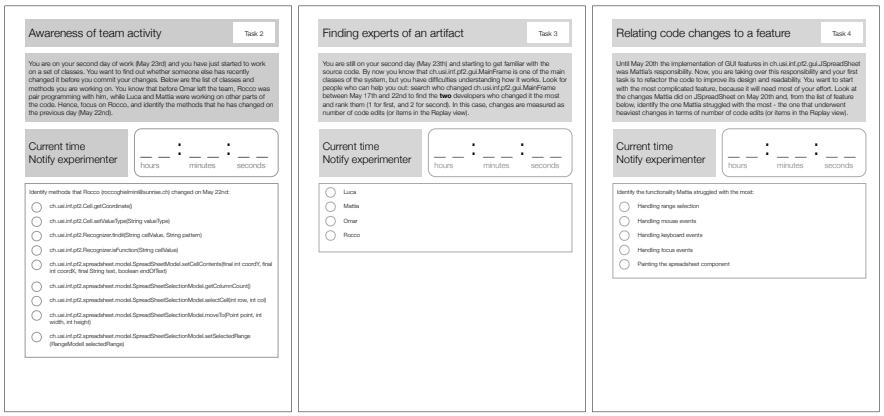


Fig. 10: Handout of treatment T2 (experimental group) – Part 1 of 2

Fig. 11: Handout of treatment T2 (experimental group) – Part 2 of 2

Imagine that you are on May 22nd and that you are joining this team to replace Omar, who was allocated to another project. This company loosely follows a software process, so Omar did not document what he was doing before he left. Your first task is to find out what he was working on, so you can start from what he left unfinished. Look at the changes Omar has done during the week (May 17th to 21st) and identify the two classes he changed the most. Changes should be quantified with number commits per class.

The two classes Omar (elabedomar) changed the most between May 17th and May 21st are:

1. class .....
2. class .....

Choices:
All classes of the system.

Solution:

- ch.usi.inf.pf2.gui.JSpreadSheet
- ch.usi.inf.pf2.spreadsheet.model.SpreadSheetSelectionModel

(0.5p for each correct class)

Task 2: Awareness of team activity

You are on your second day of work (May 23rd) and you have just started to work on a set of classes. You want to find out whether someone else has recently changed it before you commit your changes. Below are the list of classes and methods you are working on. You know that before Omar left the team, Rocco was pair programming with him, while Luca and Mattia were working on other parts of the code. Hence, focus on Rocco, and identify the methods that he has changed on the previous day (May 22nd).

Identify methods that Rocco (roccoghielmini@sunrise.ch) changed on May 22nd.

Choices:

- ch.usi.inf.pf2.Cell.getCoordinate()
- ch.usi.inf.pf2.Cell.setValueType(String valueType)
- ch.usi.inf.pf2.Recognizer.findit(String cellValue, String pattern)
- ch.usi.inf.pf2.Recognizer.isFunction(String cellValue)
- ch.usi.inf.pf2.spreadsheet.model.SpreadSheetModel.setCellContents(final int coordY, final int coordX, final String text, boolean endOfText)
- ch.usi.inf.pf2.spreadsheet.model.SpreadSheetSelectionModel.getColumnCount()
- ch.usi.inf.pf2.spreadsheet.model.SpreadSheetSelectionModel.selectCell(int row, int col)
- ch.usi.inf.pf2.spreadsheet.model.SpreadSheetSelectionModel.moveTo(Point point, int width, int height)
- ch.usi.inf.pf2.spreadsheet.model.SpreadSheetSelectionModel.setSelectedRange(RangeModel selectedRange)

Solution:

- ch.usi.inf.pf2.Recognizer.isFunction(String cellValue)
- SpreadSheetModel.setCellContents(final int coordY, final int coordX, final String text, boolean endOfText)
- ch.usi.inf.pf2.spreadsheet.model.SpreadSheetSelectionModel.selectCell(int row, int col)

(0.33p for each correct method)

Task 3: Finding experts of an artifact

You are still on your second day (May 23th) and starting to get familiar with the source code. By now you know that ch.usi.inf.pf2.gui.MainFrame is one of the main classes of the system, but you have difficulties understanding how it works. Look for people who can help you out: search who changed ch.usi.inf.pf2.gui.MainFrame between May 17th and 22nd to find the two developers who changed it the most and rank them (1 for first, and 2 for second). In this case, changes are measured as number of lines of code changes (added/deleted).

Choices:
All developers.

Solution:

1. Luca (lucaurso)
2. Mattia (mattia.candeloro89)

(0.5p for each correct class)

Task 4: Relating code changes to a feature

Until May 20th the implementation of GUI features in ch.usi.inf.pf2.gui.JSpreadSheet was Mattias responsibility. Now, you are taking over this responsibility and your first task is to refactor the code to improve its design and readability. You want to start with the most complicated feature, because it will need most of your effort. Look at the changes Mattia did on JSpreadSheet on May 20th and, from the list of feature below, identify the one Mattia struggled with the most – the one that underwent heaviest changes in terms of number of lines of code changes (added/deleted).

Choices:

- Handling range selection
- Handling mouse events
- Handling keyboard events
- Handling focus events
- Painting the spreadsheet component

Solution:

Handling keyboard events (1p)

Task 5: Tracking back the introduction of a bug

You have been working with the team for one week now and your next task is to fix a bug. The bug happens when someone tries to open an existing '.csv' file on the spreadsheet. To reproduce the bug, find class ch.usi.inf.pf2.gui.MainGUI, right-click on it and run as 'Java Application'. Then, click on 'Open', and select 'test.csv'. You should get the exception illustrated below. Rocco told you that the bug wasn't happening with him until last time he changed the code (on May 27th). Based on this information, answer the following questions.

(An image with the exception is shown)

a. When was the bug introduced?

Choices:

Entire period of the development of the system.

Solution:

30.05.2010

b. Who introduced the bug?

Choices:

All developers of the system.

Solution:

Luca

c. What change caused the bug?

Solution:

In free form saying that what caused the bug was the change of the call sheet.getGrid().get(i).add(Cell) to sheet.getGrid().get(i).add(int,Cell).

d. Propose a fix to the bug

Solution:

There are a couple of possibilities, but the easier one is to revert to the old code.

(0.25p for each correct answer)

Task 6: Understanding past refactorings

This system is now on its maintenance phase. The other developers were allocated to new projects and you are maintaining it alone. Since you joined the team almost at the end of the development cycle, when everything is very hectic, you didn't have time to deeply understand the system's architecture. Now, investigating the code, you've noticed that right before you joined the team (around May 17^{th} to 21^{st}) a major refactoring took place, involving the deletion of class ch.usi.inf.pf2.Sheet. This refactoring was mainly done by Luca with small contributions by the other developers. Investigate why Luca removed ch.usi.inf.pf2.Sheet, and what other classes were changed in the same refactoring. Describe the refactoring, giving details about what classes changed and why.

Solution:
Subjective answer: does not count for the qualitative analysis.

A.3.3 Debriefing Questionnaire

Time pressure. On a scale from 1 to 5, how did you feel about the time pressure? Please write in the box below the answer that matches your opinion the most:

1. Too much time pressure. I could not cope with the tasks, regardless of their difficulty
2. Fair amount of pressure. I could certainly have done better with more time.
3. Not so much time pressure. I had to hurry a bit, but it was ok
4. Very little time pressure. I felt quite comfortable with the time given
5. No time pressure at all

Difficulty. Regardless of the given time, please indicate how difficult would you rate the tasks? Please mark the appropriate difficulty for each of the tasks.

Scale:
1 – trivial; 2 – simple; 3 – intermediate; 4 – difficult; 5 – impossible.

Realism. How realistic were the tasks? Please indicate how much you agree that the tasks were realistic (you can see the situation happening in real development scenario).

Scale:
1 – strongly disagree; 2 – disagree; 3 – undecided; 4 – agree; 5 – strongly agree.

Comments on the experiment. Enter comments and/or suggestions you may have about the experiment, which could help us improve it.

Comments on the Replay tool. Enter comments and/or suggestions to improve Replay (applicable for the experimental group).

A.4 Dataset

To provide a fully transparent experimental setup, we make available the entire data set of our experiment.

In Table 7 we present the subjects and the personal information that we relied on when we assigned them to the different blocks (*i.e.,* based on experience and background). Once the subjects were assigned to the two blocks (beginner/advanced), within each block we assigned the subjects to a treatment using randomization. The assignment of subjects to treatments and blocks is also presented in Table 7.

The correctness level per task for each subject is presented in Table 8. Similarly, the completion times for each tasks and overall are presented in Table 9. Finally, Table 10 and tab:realism present the data we collected from the subjects regarding the perceived time pressure, difficulty, and realism per task, as experienced by our subjects. This data allowed us to determine whether there was a task which was highly unfair for one of the groups. Moreover, it provided us important hints on the type of tasks in which Replay is most beneficial and for which type of users.

Tab. 7: The subjects personal information, clustered by treatment combination (control/experimental)

					Experience level			Years of experience		
Subject ID	**Treatment**	**Block**	**Age**	**Job position**	**Java**	**Eclipse**	**SVN**	**Java**	**Eclipse**	**SVN**
E1	Ecli+Replay	advanced	27	PhD student	knowledgeable	knowledgeable	advanced	4 – 6	4 – 6	4 – 6
E2	Ecli+Replay	advanced	24	Master student	expert	expert	expert	4 – 6	4 – 6	4 – 6
E3	Ecli+Replay	advanced	23	Master student	knowledgeable	knowledgeable	knowledgeable	4 – 6	4 – 6	4 – 6
E4	Ecli+Replay	beginner	27	PhD student	knowledgeable	knowledgeable	beginner	4 – 6	1 – 3	< 1
E5	Ecli+Replay	advanced	53	Professor	advanced	advanced	advanced	> 10	7 – 10	7 – 10
E6	Ecli+Replay	advanced	31	PhD student	advanced	advanced	advanced	> 10	7 – 10	7 – 10
E7	Ecli+Replay	beginner	27	PhD student	expert	knowledgeable	advanced	7 – 10	1 – 3	1 – 3
E8	Ecli+Replay	beginner	30	PhD student	advanced	advanced	knowledgeable	1 – 3	4 – 6	< 1
E9	Ecli+Replay	beginner	29	PhD student	beginner	beginner	beginner	1 – 3	1 – 3	< 1
E10	Ecli+Replay	advanced	29	PhD student	advanced	advanced	advanced	7 – 10	4 – 6	4 – 6
E11	Ecli+Replay	beginner	27	PhD student	knowledgeable	knowledgeable	knowledgeable	1 – 3	1 – 3	1 – 3
E12	Ecli+Replay	beginner	27	PhD student	knowledgeable	knowledgeable	none	4 – 6	4 – 6	< 1
E13	Ecli+Replay	advanced	30	PhD student	expert	expert	expert	7 – 10	7 – 10	4 – 6
E14	Ecli+Replay	advanced	26	PhD student	advanced	advanced	advanced	4 – 6	4 – 6	4 – 6
E15	Ecli+Replay	beginner	23	Master student	advanced	advanced	knowledgeable	1 – 3	1 – 3	1 – 3
E16	Ecli+Replay	beginner	30	Master student	advanced	knowledgeable	knowledgeable	1 – 3	1 – 3	1 – 3
E17	Ecli+Replay	beginner	27	Master student	advanced	knowledgeable	none	7 – 10	4 – 6	< 1
E18	Ecli+Replay	advanced	23	Master student	expert	advanced	knowledgeable	4 – 6	4 – 6	1 – 3
E19	Ecli+Replay	beginner	25	Master student	advanced	knowledgeable	beginner	4 – 6	1 – 3	1 – 3
E20	Ecli+Replay	beginner	26	Master student	knowledgeable	knowledgeable	beginner	1 – 3	1 – 3	1 – 3
C1	Ecli+SVN	beginner	25	PhD student	beginner	beginner	none	< 1	< 1	< 1
C2	Ecli+SVN	advanced	25	Master student	knowledgeable	knowledgeable	beginner	7 – 10	7 – 10	1 – 3
C3	Ecli+SVN	beginner	27	PhD student	advanced	advanced	knowledgeable	4 – 6	1 – 3	1 – 3
C4	Ecli+SVN	advanced	34	Post-doc	advanced	advanced	knowledgeable	> 10	4 – 6	7 – 10
C5	Ecli+SVN	beginner	26	PhD student	knowledgeable	beginner	beginner	< 1	< 1	< 1
C6	Ecli+SVN	beginner	29	PhD student	knowledgeable	knowledgeable	beginner	4 – 6	4 – 6	< 1
C7	Ecli+SVN	advanced	26	Master student	advanced	advanced	knowledgeable	7 – 10	7 – 10	4 – 6
C8	Ecli+SVN	beginner	25	PhD student	advanced	knowledgeable	none	7 – 10	4 – 6	< 1
C9	Ecli+SVN	beginner	30	PhD student	beginner	beginner	beginner	< 1	< 1	< 1
C10	Ecli+SVN	advanced	29	PhD student	expert	expert	expert	7 – 10	4 – 6	1 – 3
C11	Ecli+SVN	advanced	27	PhD student	advanced	knowledgeable	knowledgeable	7 – 10	7 – 10	4 – 6
C12	Ecli+SVN	beginner	33	PhD student	advanced	knowledgeable	knowledgeable	4 – 6	1 – 3	1 – 3
C13	Ecli+SVN	advanced	30	PhD student	expert	expert	expert	7 – 10	7 – 10	4 – 6
C14	Ecli+SVN	advanced	38	PhD student/Assistant Prof.	advanced	advanced	knowledgeable	4 – 6	4 – 6	4 – 6
C15	Ecli+SVN	advanced	31	PhD student	expert	advanced	advanced	4 – 6	4 – 6	1 – 3
C16	Ecli+SVN	beginner	33	Master student	beginner	beginner	beginner	1 – 3	1 – 3	< 1
C17	Ecli+SVN	beginner	22	Master student	none	beginner	none	< 1	< 1	< 1
C18	Ecli+SVN	beginner	23	Master student	advanced	advanced	knowledgeable	4 – 6	1 – 3	4 – 6
C19	Ecli+SVN	beginner	27	Master student	knowledgeable	advanced	beginner	1 – 3	1 – 3	< 1
C20	Ecli+SVN	beginner	22	Master student	knowledgeable	knowledgeable	none	4 – 6	1 – 3	< 1

Tab. 8: The correctness of the subjects' solutions to the tasks

	Correctness per task					
Subject ID	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**	**Total**
E1	1.00	1.00	1.00	1.00	0.50	4.50
E2	1.00	1.00	1.00	0.00	1.00	4.00
E3	1.00	1.00	1.00	1.00	0.50	4.50
E4	1.00	0.67	1.00	1.00	0.00	3.67
E5	1.00	1.00	1.00	1.00	0.00	4.00
E6	1.00	1.00	1.00	1.00	1.00	5.00
E7	1.00	1.00	1.00	1.00	1.00	5.00
E8	1.00	1.00	1.00	1.00	1.00	5.00
E9	1.00	1.00	1.00	1.00	0.50	4.50
E10	1.00	1.00	1.00	1.00	1.00	5.00
E11	1.00	1.00	1.00	1.00	1.00	5.00
E12	1.00	0.67	1.00	1.00	0.00	3.67
E13	1.00	0.67	1.00	1.00	0.25	3.92
E14	1.00	1.00	1.00	0.00	1.00	4.00
E15	1.00	1.00	1.00	0.00	0.00	3.00
E16	1.00	1.00	1.00	1.00	0.00	4.00
E17	1.00	0.33	1.00	0.00	0.00	2.33
E18	1.00	1.00	1.00	1.00	1.00	5.00
E19	1.00	0.67	1.00	0.00	0.50	3.17
E20	1.00	1.00	1.00	0.00	0.00	3.00
C1	0.50	0.67	1.00	0.00	0.50	2.67
C2	1.00	1.00	1.00	1.00	0.50	4.50
C3	1.00	1.00	0.00	0.00	1.00	3.00
C4	1.00	1.00	0.00	1.00	1.00	4.00
C5	1.00	1.00	1.00	1.00	0.75	4.75
C6	1.00	0.67	1.00	1.00	0.00	3.67
C7	1.00	1.00	0.00	1.00	0.75	3.75
C8	1.00	1.00	1.00	1.00	0.00	4.00
C9	0.00	1.00	1.00	1.00	1.00	4.00
C10	1.00	0.67	1.00	0.00	1.00	3.67
C11	0.50	1.00	1.00	1.00	1.00	4.50
C12	0.00	1.00	1.00	1.00	0.75	3.75
C13	1.00	1.00	1.00	1.00	1.00	5.00
C14	1.00	1.00	1.00	1.00	0.25	4.25
C15	0.00	0.67	0.50	1.00	1.00	3.17
C16	1.00	0.00	0.50	0.00	0.00	1.50
C17	0.50	1.00	1.00	1.00	0.00	3.50
C18	1.00	0.67	1.00	1.00	0.00	3.67
C19	0.00	0.00	1.00	0.00	0.00	1.00
C20	0.00	1.00	1.00	0.00	0.00	2.00

Tab. 9: The subjects' completion time per tasks (in minutes)

	Completion time per task							
Subject ID	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**	**Task 6**	**Total**	**Total (excl. T6)**
E1	4.00	4.00	1.00	7.00	18.00	17.00	51.00	34.00
E2	4.83	6.33	1.83	2.75	22.42	22.33	60.50	38.17
E3	3.08	7.33	1.67	16.42	25.42	0.00	53.92	53.92
E4	5.50	5.33	2.50	10.08	23.93	11.33	58.68	47.35
E5	9.75	3.25	1.17	9.92	20.67	21.92	66.67	44.75
E6	4.67	5.50	2.25	7.42	15.00	8.08	42.92	34.83
E7	4.75	8.75	2.58	7.50	19.33	8.83	51.75	42.92
E8	7.00	7.00	4.00	11.00	14.00	20.00	63.00	43.00
E9	6.00	7.00	4.00	8.00	20.00	28.00	73.00	45.00
E10	4.00	7.00	2.00	9.00	22.00	50.00	94.00	44.00
E11	2.00	6.00	1.00	8.00	15.00	19.00	51.00	32.00
E12	5.00	7.00	4.00	8.00	20.00	13.00	57.00	44.00
E13	3.08	6.75	1.20	5.25	20.83	20.17	57.28	37.12
E14	6.00	9.00	3.00	7.00	20.00	19.00	54.00	38.00
E15	3.58	6.17	2.08	6.00	23.00	22.00	63.17	40.83
E16	2.25	9.58	2.42	6.92	19.67	24.33	65.17	40.83
E17	9.50	6.42	4.67	6.53	21.03	23.83	71.98	48.15
E18	5.33	7.00	4.25	6.92	18.75	13.33	55.58	42.25
E19	4.50	7.00	4.33	10.25	20.00	28.25	74.33	46.08
E20	6.08	6.75	2.25	4.25	20.33	26.00	65.67	39.67
C1	5.00	4.00	13.00	3.00	14.00	12.00	51.00	39.00
C2	8.33	5.75	10.33	10.75	14.33	27.00	76.50	49.50
C3	7.00	7.00	9.00	10.00	21.00	18.00	72.00	54.00
C4	7.00	10.00	9.00	8.00	16.00	28.00	78.00	50.00
C5	9.18	3.42	10.67	1.67	20.25	14.50	59.68	45.18
C6	7.33	3.77	4.25	5.92	23.92	25.42	70.60	45.18
C7	6.08	5.50	10.00	7.67	21.33	16.42	67.00	50.58
C8	8.00	9.00	10.00	5.00	19.00	15.00	66.00	51.00
C9	9.00	10.00	9.00	5.00	17.00	19.00	69.00	50.00
C10	7.92	6.10	9.83	4.25	23.00	13.50	64.60	51.10
C11	6.92	6.67	7.83	4.83	14.67	23.00	63.92	40.92
C12	7.25	8.00	11.50	6.08	11.00	21.33	65.17	43.83
C13	4.37	7.58	6.67	4.00	13.58	10.50	46.70	36.20
C14	4.00	6.00	3.00	4.00	20.00	21.00	58.00	37.00
C15	6.00	5.00	6.00	5.00	16.00	16.00	54.00	38.00
C16	11.33	9.00	13.75	9.17	12.17	13.33	68.75	55.42
C17	10.50	7.83	9.33	5.25	19.92	8.50	61.33	52.83
C18	8.42	4.33	5.83	5.33	21.58	14.57	60.07	45.50
C19	11.67	10.17	4.95	7.83	20.83	27.75	83.20	55.45
C20	12.50	9.00	7.25	9.42	17.38	17.25	72.80	55.55

Tab. 10: The subjects' perceived time pressure and task difficulty

Subject ID	Time pressure	Difficulty level					
		Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
E1	3	simple	simple	simple	simple	intermediate	difficult
E2	3	trivial	intermediate	simple	intermediate	simple	difficult
E3	1	simple	simple	simple	intermediate	difficult	
E4	3	trivial	simple	trivial	intermediate	difficult	difficult
E5	2	trivial	simple	simple	intermediate	impossible	difficult
E6	4	intermediate	simple	intermediate	intermediate	difficult	difficult
E7	3	trivial	simple	trivial	intermediate	intermediate	impossible
E8		difficult	difficult	simple	impossible	difficult	impossible
E9	2	simple	simple	simple	simple	intermediate	difficult
E10	2	trivial	simple	trivial	simple	simple	difficult
E11	3	trivial	simple	simple	simple	intermediate	difficult
E12	2	difficult	difficult	difficult	difficult	difficult	difficult
E13	2	trivial	simple	simple	intermediate	difficult	difficult
E14	4	trivial	simple	trivial	intermediate	difficult	difficult
E15	4	simple	simple	simple	simple	difficult	intermediate
E16	2	trivial	simple	trivial	simple	intermediate	difficult
E17	2	trivial	trivial	simple	intermediate	difficult	difficult
E18	4	simple	trivial	trivial	intermediate	intermediate	difficult
E19	2	trivial	simple	simple	intermediate	difficult	impossible
E20	3	simple	simple	trivial	intermediate	difficult	difficult
C1	3	simple	simple	intermediate	intermediate	intermediate	intermediate
C2	4	simple	simple	difficult	difficult	intermediate	intermediate
C3	2	trivial	trivial	simple	intermediate	simple	difficult
C4	3	trivial	trivial	trivial	simple	intermediate	difficult
C5	3	intermediate	simple	difficult	intermediate	difficult	simple
C6	3	simple	trivial	intermediate	intermediate	difficult	intermediate
C7	2	simple	trivial	impossible	difficult	intermediate	intermediate
C8	2	intermediate	simple	intermediate	intermediate	impossible	intermediate
C9	3	simple	simple	intermediate	intermediate	difficult	intermediate
C10	3	trivial	trivial	intermediate	intermediate	simple	simple
C11	3	intermediate	simple	difficult	difficult	intermediate	difficult
C12	5	simple	simple	difficult	intermediate	simple	simple
C13	2	intermediate	simple	difficult	intermediate	simple	intermediate
C14	2	trivial	trivial	trivial	simple	difficult	difficult
C15	4	simple	simple	simple	simple	intermediate	difficult
C16	1	intermediate	difficult	impossible	difficult	difficult	impossible
C17	4	intermediate	simple	intermediate	trivial	difficult	difficult
C18	3	simple	simple	intermediate	simple	intermediate	difficult
C19	1	intermediate	intermediate	simple	simple	difficult	difficult
C20	2	difficult	difficult	intermediate	intermediate	intermediate	undecided

Tab. 11: The subjects' perceived realism of each task

Subject	Task realism					
ID	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
E1	disagree	disagree	agree	undecided	agree	strongly agree
E2	agree	agree	agree	agree	undecided	agree
E3	agree	agree	agree	agree	agree	
E4	undecided	undecided	undecided	undecided	undecided	undecided
E5	agree	agree	agree	agree	agree	agree
E6	undecided	agree	agree	undecided	strongly agree	strongly agree
E7	disagree	disagree	agree	strongly disagree	strongly agree	agree
E8	undecided	undecided	agree	disagree	agree	strongly agree
E9	agree	agree	agree	agree	agree	agree
E10	agree	strongly agree	agree	strongly agree	strongly agree	agree
E11	undecided	undecided	agree	agree	agree	agree
E12	agree	agree	agree	agree	agree	agree
E13	strongly agree	strongly agree	strongly agree	undecided	agree	agree
E14	agree	agree	strongly agree	undecided	agree	agree
E15	agree	undecided	agree	agree	strongly agree	undecided
E16	agree	strongly agree	strongly agree	agree	strongly agree	strongly agree
E17	agree	agree	agree	undecided	strongly agree	strongly agree
E18	agree	disagree	undecided	agree	strongly agree	strongly agree
E19	disagree	disagree	undecided	agree	strongly agree	strongly agree
E20	agree	strongly agree	strongly agree	undecided	strongly agree	undecided
C1	agree	agree	agree	agree	agree	agree
C2	undecided	agree	agree	undecided	agree	agree
C3	agree	undecided	agree	agree	strongly agree	disagree
C4	agree	agree	disagree	strongly disagree	agree	strongly disagree
C5	disagree	disagree	strongly agree	undecided	strongly agree	agree
C6	strongly agree	agree	disagree	disagree	agree	strongly agree
C7	agree	agree	disagree	undecided	strongly agree	agree
C8	agree	agree	agree	agree	agree	agree
C9	undecided	undecided	agree	undecided	strongly agree	strongly agree
C10	agree	agree	strongly agree	undecided	strongly agree	strongly agree
C11	undecided	undecided	strongly agree	disagree	disagree	agree
C12	agree	agree	disagree	disagree	agree	undecided
C13	agree	agree	undecided	agree	agree	agree
C14	agree	agree	agree	undecided	strongly agree	
C15	strongly agree	strongly agree	strongly agree	undecided	agree	
C16	undecided	agree	undecided	agree	disagree	disagree
C17	disagree	agree	agree	agree	strongly agree	agree
C18	agree	agree	strongly agree	agree	strongly agree	strongly agree
C19	agree	strongly agree	undecided	undecided	strongly agree	agree
C20	undecided	undecided	undecided	undecided	agree	agree