Assessing the Effect of Requirements Traceability for Software Maintenance

Patrick Mäder and Alexander Egyed
Institute for Systems Engineering and Automation (SEA)
Johannes Kepler University, Linz, Austria
patrick.maeder|alexander.egyed@jku.at

***Abstract*—Advocates of requirements traceability regularly cite advantages like easier program comprehension and support for software maintenance (i.e., software change). However, despite its growing popularity, there exists no published evaluation about the usefulness of requirements traceability. It is important, if not crucial, to investigate whether the use of requirements traceability can significantly support development tasks to eventually justify its costs. We thus conducted a controlled experiment with 52 subjects performing real maintenance tasks on two third-party development projects: half of the tasks with and the other half without traceability. Our findings show that subjects with traceability performed on average 21% faster on a task and created on average 60% more correct solutions – suggesting that traceability not only saves downstream cost but can profoundly improve software maintenance quality. Furthermore, we aimed for an initial cost-benefit estimation and set the measured time reductions by using traceability in relation to the initial costs for setting-up traceability in the evaluated systems.**

***Keywords*-requirements traceability; software maintenance; traceability effect; empirical software engineering; experiment;**

I. INTRODUCTION

Requirements-to-code traceability reflects the knowledge where requirements are implemented in the code and its capture and maintenance is the focus of extensive research. Despite its growing popularity [1] surprisingly little is known about its benefits. Intuitively, requirements-to-code traces should be useful for many areas of software engineering. Authors refer to better program comprehension and support for software maintenance (e.g., [2], [3]). In the safety critical domain, traceability supports the certification process and is thus mandated by several standards (e.g., DO-178B and EIA 12207). Nonetheless, there exists no empirical work in which the effect of requirements traceability was proofed and measured.

The first goal of this work was to investigate whether knowledge of requirements-to-code traces improves the performance of subjects during software maintenance tasks. Software maintenance is an important part of the development process, where developers are no longer as familiar with the code as during the initial development, and traces are perceived to be useful. To study that effect, we carried out a study involving 52 students with a wide range of experiences. The subjects were asked to solve maintenance tasks taken from two software development efforts: the open source Gantt Project (47 KLOC) and the iTurst system (15 KLOC). Eight tasks were selected, covering real bug fixes and feature extensions taken from their documented archives (Gantt: issue tracker, iTrust: wiki with requirements specification). Tasks were randomly assigned to subjects, half the tasks with and the other half without traceability. We measured the performance of subjects as the time they spent to solve a task and the correctness of their solution. Having selected real maintenance tasks provided us with a golden standard as to how their original developers solved the given tasks. This allowed us to assess the correctness of the solutions the subjects provided for the tasks.

The second goal of this work was to characterize performance differences due to traceability in relation to a range of other criteria. We assessed how the performance varies based on the kind of tasks subjects solved, the different domains that projects came from, and the order in which a task was performed by a subject. All subjects had in common that they were not familiar with the applied projects – a situation that commonly occurs during software maintenance and a situation under which developers are expected to benefit from requirements traceability.

In total, subjects were solving 315 tasks (i.e., 6 tasks per subject on average). Our findings show that subjects working on tasks with traceability performed better than subjects working without traceability. In particular, subjects with traceability performed on average 21% faster on tasks and created on average 60% more correct solutions. This demonstrates that traceability is not just a means for saving some effort but can profoundly improve the quality of the software maintenance process with likely many subsequent benefits such as more effective maintenance, faster time to market, and less code degradation. We also found that some tasks benefited more from traceability than others, especially regarding correctness of the solution. Furthermore, we found that though project domain does significantly affect the performance of subjects, it did not impact the effect of traceability. The former is particularly surprising because the two systems differed in their characteristics.

The implications of this study are numerous. Traceability strongly benefits software maintenance. While the capture and maintenance of traceability (studied in other works [4]) does require significant effort and while developers tend to perceive this activity to be tedious and ineffective [5], this work nonetheless demonstrates a clear, measurable performance improvement to justify this cost. Since this work clearly characterizes the effect of traceability, practitioners and researchers alike may use this information to better understand the cost/benefit trade-off of traceability – a point that will be the focus of our future work.

The remainder of this paper is structured as follows, Section II discusses related studies in the area of requirements traceability and software engineering. Section III introduces our experimental set-up, while Section IV reports about the results and findings of the experiment. In Section V we discuss the limitations of that study and in Section VI we roughly estimate how measured benefits relate to the cost of requirements traceability. Finally, Section VII concludes and discusses possible areas of future investigations.

II. Related Work

Gotel and Finkelstein report the findings of a year-long empirical study into traceability conducted in 1992 [6]. The study involved around one hundred software development practitioners, holding a variety of positions within a large organization, with experience of up to 30 years. In addition to a comprehensive questionnaire, five focus group sessions were conducted with thirty-seven practitioners to consolidate data, and independent observation of practical requirements gathering and development workshops took place. The authors found multiple perspectives on what traceability was expected to enable and on the problems experienced, and conflicts particularly evident between those parties responsible for establishing traceability and those parties using it. The authors were concerned with understanding and exposing the scope of the problem area; they did not report about the actual benefits of traceability to their subjects.

Ramesh and Jarke report on a large practitioner study of traceability where the data collection took over three years during the 1990's [7]. The authors conducted a pilot study with fifty-eight masters in information technology students to create an initial traceability meta-model and inform the design of the study. The main study consisted of thirty focus group discussions, each with about five people from twenty-six companies. Their primary focus was on the types of traceability link used in current and ideal practice. The study comprised two phases: the first revolved around agreeing a traceability meta-model and the second on defining reference models for other practitioners to use. Again, the actual benefits of traceability to their subjects were not exposed.

Arkley and Riddle report on a survey of nine software projects, small to multinational in scope, undertaken using questionnaires and interviews [5]. The authors identified three issues related to traceability: the necessity for extra entry data when using traceability tools; a lack of understanding on how to employ traceability; and the lack of perceived direct benefits to the main development process.

Ahmad and Ghazali interviewed fifteen practitioners and analyzed project documentation of three IT companies [8]. Their subjects had six to ten years of practical experience in developing small projects. Their finding was that subjects perceived pre-requirements traceability to be more beneficial than post-requirements traceability. The authors did not study the benefits of traceability to their subjects in detail.

Additionally, there is a variety of authors evaluating the effect of different traceability techniques, e.g., [9]–[11], but none of them is evaluating the effect of traceability itself. Traces are rarely perfect but to full potential of traceability, we presume correctness and completeness (see Section V).

Based on the overview of empirical studies in the area, we identify a lack of work assessing the effect of traceability for development tasks. Following the argumentation in the introduction, maintenance is a major cost driver in the development of a system [12] and traceability could reduce that costs. Accordingly, we decided to restrict our focus to software maintenance. We found several studies focusing on software maintenance, e.g., Dzidek et al. [13] study the costs and benefits of UML documentation for software maintenance and Curtis et al. [14] compare the performance of subjects solving maintenance tasks with complexity measures (e.g., Halstead and McCabe metrics) evaluating the same tasks.

III. Method

A controlled experiment was conducted in which participants had to perform several maintenance tasks with and without traceability. The research investigated the effort and quality differences of participants and assessed whether traceability significantly impacted it.

A. Participants

The participants comprised 52 students of computer science, studying at the JKU Linz. We required all subjects to have at least basic experience of software development in general and development with JAVA in particular. The participants had an average experience of 5.3 years in software development and an average experience of 3.4 years in development with JAVA. On average, each student had developed software in industrial environments for 1 year, though some had no industrial experience at all and others up to 6 years. None of the participants had previous knowledge of the application's source code used for the experiment. Only four of the participants had previously used traceability.

B. Independent Variables

Our main interest was to assess whether traceability can have a significant impact on the performance of a software maintainer if exposed to an unknown software system. In order to ensure generalizable results, we decided to use two different software projects with different characteristics and to request participants to perform four different tasks per project (removing bugs and implementing change requests). Because we expected subjects to learn about a project with every task (carryover or learning effect), we considered the order in which a task appeared. All these distinctions were considered as independent variables in our experiment and subsequent statistical evaluation. The following paragraphs discuss each variable in depth.

Project (P): We selected tasks from two software systems: Gantt and iTrust in order to ensure generalizability of results. GanttProject is an open-source, cross-platform tool for project scheduling, implemented with JAVA. The tool generates Gantt charts that breakdown the work structure of a project into tasks, dependencies among tasks, and milestones [15]. All the functionality of the tool deals with these charts. We selected GanttProject for our experiment, because it represents a class of development project where documentation is kept to a minimum and traceability is less likely to be created and used. One of the key developers of the GanttProject provided the necessary requirements traces. Their issue tracking system and source code repository keeps track of changes and allows for a detailed determination of bug reports and resulting code changes.

Table I
DESCRIPTIVE STATISTICS AND METRICS OF THE SOFTWARE PROJECTS

Metric	Gantt	iTrust
Total lines of code (TLOC)	47k	15.5k
Number of classes (NOC)	626	232
Number of methods (NOM)	4570	1612
Avg. depth of inheritance tree (DIT)	2.097	1.272
Max. depth of inheritance tree (DIT)	8	4
Cyclomatic complexity (VG)	1.697	1.553

iTrust is a web-based medical application that provides patients and other medical stakeholders with a means to keep up with their medical records as well as to communicate between each other. iTrust is implemented in JAVA (java server pages for user interface) and is being developed by Laurie Williams and her students for several years [16]. Though not an industrial project, iTrust is a complete application developed and improved in eleven major versions. iTrust has been developed according to a regulatory code and is the type of system that usually employs traceability. iTrust is a well structured and documented system, in accordance to design notes and HIPPA regulations.

Table I shows metrics of the two projects. Both have industrial size, though Gantt is roughly three times larger than iTrust. The iTrust code is well structured and the extensive use case descriptions are likely to ensure a consistent problem understanding through subjects. In contrast, metrics show that the Gantt code is harder to maintain (Table I) and the short and less precise bug reports may be easier to understand for some subjects than for others.

Table II
OVERVIEW OF TASKS FOR BOTH PROJECTS AND THE NUMBER OF PROVIDED TRACEABILITY LINKS FOR EACH

Task		Issue or change request	Traces
Gantt	A	[Issue id: 1755404]: New, unsaved chart discarded after canceling *Save* (select folder, filename) dialogue box	5
	B	[Issue ids: 1018957, 1115471]: Finish-Finish relations are saved as Finish-Start relations	2
	C	[Issue id: 1364493]: *Save* works as *Save As* for new project: file chooser dialogue appears again, though the project has been properly saved before	5
	D	[Issue id: 2006796]: Only project files with lowercase **.gan* filename extensions are accepted under Windows	8
iTrust	A	[View emergency electronic health record (UC21), ver 14.1, change: Oct. 22, 2008]: list prescription as current prescription if OLD:{prescribed within the last 90 days} $\rightarrow$ NEW:{end date of the prescription is within the last 91 days}	18
	B	[View records (UC9), ver 13, change: Oct. 15, 2008]: add information whether family members suffer(ed) from heart disease	12
	C	[Proactively Determine Needed Patient Care (UC17), ver 14, change: Oct. 19, 2008]: add Poliovirus immunization when checking for needed immunizations	7
	D	[View Access Log (UC8), ver 16, change: Sep 7–16, 2009]: add functionality to sort access log by role of accessor	4

Task (Tk): altogether, our experiment included eight distinct tasks: four tasks for Gantt (subsequently referred to as tasks Gantt A to D) and four tasks for iTrust (iTrust A to D). The tasks represent maintenance activities that previously occurred in these projects (see Table II). For Gantt we selected bug reports from the issue tracking system (see Table II) and provided them unchanged to the participants. For iTrust we compared old versions of the use case specification with the current one and selected single change requests (see Table II). iTrust tasks contained the original use case description and highlighted the required change or extension. For each task, the table also contains the number of available traces for the treatment with traceability. Participants were not required to actually implement changes, but to identify the artifacts to be changed and to describe required changes in a structured questionnaire. You will find more details on tasks, acceptable solutions, the questionnaire, and the experimenting tool in the replication package at http://www.sea.jku.at/tools/.

Traceability (Tr): Traces were either available to a subject on a particular task (with traces) or were not available (no traces). Traces were provided as they had been created by the original developers, some of them tracing to a whole source code file (all Java server pages) and the remaining ones to methods within files (all JAVA code). iTrust tasks consisted of full use case descriptions with parts to be changed highlighted in the text. For tasks with traces, all available traces for a use case were provided to the subject. Traces were labeled with the name of the use case scenario and the source code element they related to. Gantt tasks consisted of a bug description to be fixed. For Gantt tasks with traces, we provided traces relating to the feature(s) impacted by the bug to be fixed. The Gantt developers had created these traces between a feature list and the source code. Gantt traces were labeled with the feature name and the related source code element name. We thoroughly checked all traces for correctness and completeness with respect to our asks and found no issues. The number of available traces per task is shown in Table II.

Task Order (O): Tasks were assigned to subjects in different orders to avoid bias. As a variable, we thus captured whether or not a task was presented to a subject as her/his first or later task per project.

C. Dependent Variable

This study had one dependent variable, the performance of subjects working on a maintenance task, which has been operationalized by two measures: time and correctness.

The time to solve a task was measured in seconds by the experimenting tool. We decided to measure correctness as a factor with three levels: incorrect, partly correct, and fully correct. All of the provided solutions were scored by the same two graders. Since our tasks were based on real changes, we knew about the correct solution chosen by the original developers, but we also accepted alternative solutions, if they lead to an acceptable result. For bugs, we considered a solution correct if it removed the problem. For feature extensions, a solution was considered correct if the desired additional functionality was provided. Thus, the performance measures of our experiment were the level of correctness that a solution had and the number of seconds required to provide that solution.

D. Experimental Design

In order to control for individual differences in performance, the experiment employed a 2x2x4x4 factorial design. Four tasks were defined for each of the two projects, each task was presented in one of four possible orders, and each task was presented with and without traceability.

Participants were randomly assigned to experimental conditions, but in such a way that they equally experienced each level of the independent variables project, task, and traceability. That is, she/he had been assigned to all four tasks of both projects, she/he either started with the four Gantt or the four iTrust tasks, and per project she/he performed two tasks with and two without provided traceability.

E. Experimenting Tool

In order to have control of what participants could do during the experiment and to capture all their actions, we implemented a specific editor tool. This ensured that all subjects would be equally treated in the experiment.

The developed tool is a text editor with development support. It provides a tree selector for browsing the project structure and opening files, multiple files can be open in separate tabs, and the content of files is being syntax highlighted (JAVA and JSP). Additionally, users can conveniently search within all files of a project and restricted to a selected file.

Traceability links are provided in a separate window above the tree selector and labeled with the names of the artifacts they are connecting (e.g., UC1[S1] $->$ GetVisitRemindersAction.java : getVisitReminders()). A click on a link opens the related file and highlights the linked artifact. Traces are also highlighted in the file tree.

Furthermore, the editor controls the experiment. Once a task is started, all required artifacts are displayed, i.e., the task description (bug report or use case description with change request), the project file tree and if available related requirements traces. The tool captures the time a user spends on a task, whether a task has been completed and tracks all the actions that the user takes to solve a task. All subjects were working with that tool for all their tasks.

F. Procedure and Material

Material: A packet of material was prepared for each participant, explaining the goal of the experiment, the nature of tasks, and how to capture results within the provided questionnaire. The material further gave a short introduction into the experimenting tool (used by all subjects regards of experiment setup) and the selected projects. For GanttProject, the material briefly described the tool and its functionality (all taken from [15]), including two screenshots of the tool. For iTrust, the material also described the functionality of the tool, provided a glossary of abbreviations used within use case scenarios, showed brief design notes and four screenshots with different views of the application (all taken from [16]). In all, the documentation for the iTrust system was more elaborate than the documentation for the GanttProject system. All material was provided in English.

Introduction: We spent 20 min going step by step through the material. The introduction further comprised instruction in the use of the experimenting tool and two practice exercises with the experimenting tool distinct from the experimental tasks. After the exercises, each participant had to complete the first part of the questionnaire, gathering information about her/his development experience, prior knowledge of the source code of our two projects and experience in using traceability (see Subsection III-A).

Tasks: The main part of the experiment contained the administration of up to eight experimental tasks. We allowed up to two hours for working on these tasks. Participants were told they had to successfully solve the tasks if possible, i.e., correctness of the solution was their goal. However, they were also required to quickly solve the problems. The

Table III
SUMMARY OF MULTIVARIATE AND UNIVARIATE ANALYSES OF VARIANCE (N=315)

		Multivariate analysis of variance for performance (time, correctness)			Univariate analyses of variance for time		correctness	
Source of variation	Df	Wilks' λ	approx. F	Pr(>F)	F value	Pr(>F)	F value	Pr(>F)
Independent variables								
traceability (Tr)	1	0.84	27.49	***	26.95	***	34.74	***
project (P)	1	0.97	4.50	*	5.32	*	4.80	*
task (Tk)	7	0.83	4.55	***	6.27	***	2.85	*
task order (O)	3	0.83	9.02	***	18.19	***	1.19	n.s.
Interactions								
Tr x Tk	7	0.92	1.76	*	1.17	n.s.	2.36	*

Significance codes for Pr(>F): > 0 '***'; > 0.001 '**'; > 0.01 '*'; > 0.025 'n.s. (non significant result)'

experimenting tool controlled the appearance of tasks (see Subsection III-D on assignment details). Once, a subject decided to stop working on a task, it was not possible for her/him to go back to that task in order to control learning effects. Participants were permitted to work up to 30 min on each task, otherwise the tool would stop the work. For each task, the participants had to capture their changes in the questionnaire that provided templates for capturing changes.

Debriefing: We required each participant to fill-in a short debriefing and feedback section of the questionnaire.

IV. RESULTS

A multivariate analysis of variance (MANOVA) was conducted to determine the effect of the independent variables on the compound dependent variable performance. Multivariate normality of the captured data was tested using a multivariate version of the Shapiro test. A Bartlett test was performed to check homogeneity of variance within the data. Both tests indicated a lack of evidence that the assumptions of normality and variance homogeneity were violated. One multivariate outlier was removed from the data set. Based on that preparations, MANOVA was considered to be an appropriate analysis technique for the given problem.

An initial statistical model describing a subject's performance, measured by time to and correctness of solution, was fitted by inclusion of all independent variables and all possible interactions between the independent variables. An interaction between two factors exists if the effect of one factor depends on the levels of the second factor. This initial model was simplified by stepwise removal of non-significant interaction terms. Table III presents the result of three model analyses, the multivariate MANOVA and the ongoing univariate ANOVAs, all computed for the simplified model.

From top to bottom, the rows of the table show statistical measures for the six independent variables as well as the two remaining interactions, all have been identified as significant sources of variation. The second column of the table shows the degree of freedom for each variable and interaction. The third to fifth columns summarize the results of the computed MANOVA. We decided to apply Wilks' λ as test statistics for assessing whether there are statistically significant differences between the independent variables and the linear combination of the dependent variables, because the quantity $1 - \lambda$ gives the proportion of generalized variance in the dependent variables explained by the model and so allows to quantify each effect. That is, the smaller the value of λ for an independent variable or interaction the more variation in the linear combination of the dependent variables it explains. The columns six, seven, eight and nine show results of two univariate analyses of variance (ANOVA) for each dependent variable that were conducted as follow-up tests to the MANOVA. Using the Bonferroni method for controlling Type I error rates for multiple comparisons, both ANOVAs were tested at a $\alpha = 0.025$ significance level. The following subsections discuss in-depth the effect of each independent variable and contain figures and tables that quantify effects.

A. Traceability and Project

This subsection discusses the effects of the independent variables traceability and project on the performance of subjects. The statistical tests show that traceability has a highly significant effect on how a subjects overall performs for a particular task (see row "traceability" of Table III: $Wilks'\lambda = 0.84, F = 27.49, ***$) as well as separately on the time she/he needs to complete the task and the correctness she/he achieves for that task. The project to which a task belongs has a less strong, but still significant effect on the result of a subject (see row "project" of Table III: $Wilks'\lambda = 0.97, F = 4.50, *$).

Figure 1 compares the times that participants spend on tasks with and without traceability as box plots for all performed tasks (left) and separately for tasks performed on the Gantt project (middle) and the iTrust project (right). Similarly, Figure 2 compares the overall correctness reached by participants and the correctness per project as bar plot. Each bar visualizes the percentages of correct solutions (dark grey area), partly correct solutions (light grey area), and

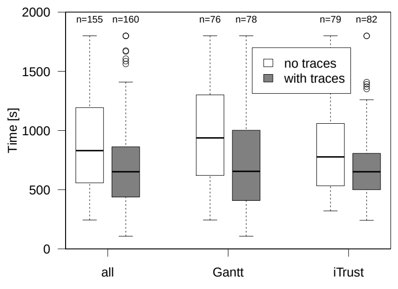

Figure 1. Time spent on performing a task across all performed tasks and per project without and with traceability

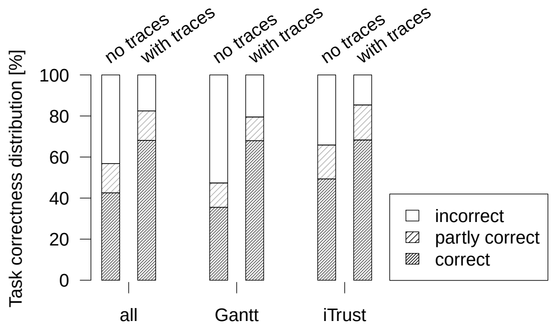

Figure 2. Correctness distribution of results across all performed tasks and per project without and with traceability

incorrect solutions (white area) to a task in comparison to all solutions. For the same three groups (all tasks, Gantt tasks, iTrust tasks), Table IV presents the mean and the standard deviation of time spent on performing a task (mean (sd)), the difference in mean values between the treatment without and with traces (diff), the percentage of correct solutions that subjects created for particular tasks (Correct tasks), and the difference in correct replies between the treatment without traces and the treatment with traces (diff).

Table IV
PERFORMANCE OF SUBJECTS ACROSS ALL TASKS AND PER PROJECT

		Time [s]		Correct tasks	
Project	Trace	mean (sd)	diff	[%]	diff
all	no	906 (411)	-21%	43	60%
	with	717 (360)		68	
Gantt	no	970 (417)	-24%	36	91%
	with	739 (413)		68	
iTrust	no	844 (398)	-18%	49	38%
	with	696 (303)		68	

A comparison of all performed tasks shows that subjects working with traceability spent on average $717s$ working on a task, while subjects working without traceability spent on average $906s$ working on a task. That means that on average traceability allowed to complete a task 21% faster. Comparing the correctness of performed tasks, we found that participants working without traceability created 43% correct solutions, while participants working with traceability created 68% correct solutions to tasks. That means that traceability facilitates 60% more correct solutions to a task.

After examining the effect that traceability has on the performance of subjects, we compared time and correctness separately for tasks performed on the Gantt and the iTrust project. Results show that iTrust tasks have been solved slightly faster and more correct than Gantt tasks. That observation correlates with the projects' metrics (see Table I) and our subjective opinion (Gantt being the larger system and being significantly less documented). That effect is statistically significant, but less strong than the effect of the other independent variables (see Table III).

We also studied the combined effect of traceability and project. Regarding time, we found that subjects working with traceability spent on average slightly more time for completing a Gantt ($739s$) vs. an iTrust task ($696s$). For subjects working without traceability the effect is similar, though the difference in times is larger, Gantt ($970s$) vs. iTrust ($844s$). Regarding correctness, subjects with traceability created 68% correct solutions for tasks of both projects. A difference is visible for subjects working without traceability, which created 36% correct solutions for Gantt tasks vs. 49% correct solutions to iTrust tasks.

In summary, the differing project characteristics of the Gantt and the iTrust project mainly effect subjects without traceability, while subjects with traceability perform almost equally (time and correctness) for both projects (compare also Figure 1 and 2). Though, this is an interesting trend, we found no significant interaction between traceability and project to statistically support that finding.

B. Tasks

This subsection focuses on the effects that the kind of task has on the performance of a subject individually and in combination with traceability. The statistical tests show that the kind of task highly significantly effects a subjects overall performance (see row "task" in Table III: $Wilks'\lambda = 0.83, F = 4.55, ***$) as well as separately on the time spent to complete the task and the achieved correctness. Figure 3 compares the times that participants spend on tasks with and without traceability separately for the eight different tasks. Similarly, Figure 4 compares the correctness that participants reached with and without traceability per task. Finally, Table V presents descriptive statistics. Especially, when comparing the results for different tasks it is important to recognize that time and correctness are dependent. Someone who is solving a task correctly may require more time than someone who fails in solving a task.

Focussing at differences in time between tasks (see Table V), subjects working with traceability completed their work, except for Gantt A, 14% to 53% faster than those subjects performing the same tasks without traceability. For the Gantt A task they performed 2% slower than those

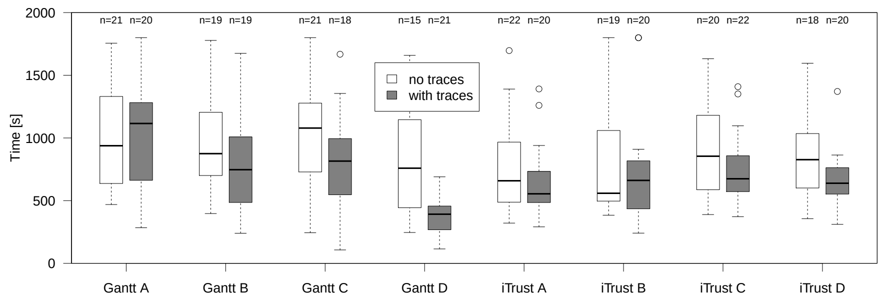

Figure 3. Time for completing a task without and with traceability, depending on its type

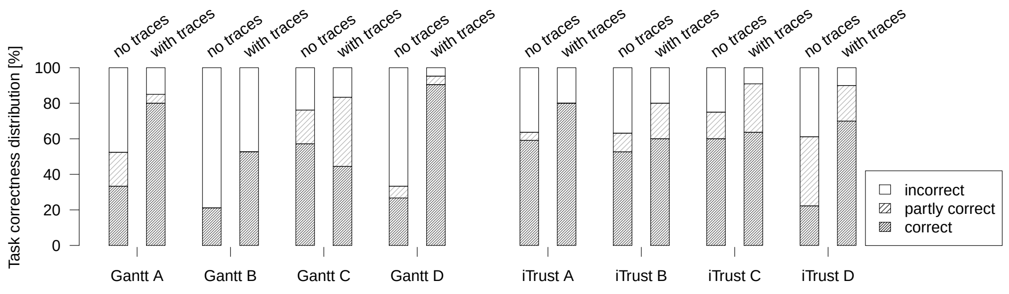

Figure 4. Correctness distribution of results created without and with available traceability, depending on the kind of task

working without traceability. The specific of the bug report used in Gantt A was that the original developers had created a multi-level, logical construct that did not properly handle the case of canceling a file selector dialog. Subjects might have spent a long time on understanding the construct in order to solve the task. Especially, if compared with correctness showing 80% vs. 33% correct solutions created for Gantt A by subjects working with traceability, we may infer that subjects without traceability used less time because most of them failed to identify the problem correctly.

Assessing correctness between the different kinds of task shows that subjects working with traceability created, except for Gantt C, 6% to 239% more fully correct solutions to tasks than those subjects working without traceability. For the Gantt C task, subjects working with traceability created 22% less fully correct solutions, while the combined percentage of partly and fully correct solutions 76% vs. 83% is larger for the group working with traceability. An analysis of that issue suggests that the five provided traces that captured the full *Save As* functionality, might have been misleading for several subjects working with traceability.

There is a wide spread in correctness differences between tasks of the same treatment (with or without traceability) and between treatments. Statistical tests identified the interaction between traceability and kind of task to significantly effect the performance of subjects (see row "Tr x Tk" in Table III: $Wilks'\lambda = 0.92, F = 1.76, *$), though not strongly significant. The univariate analysis shows that the effect restricts to correctness and does not significantly influence the time spent on the task. The finding is that the gain in speed through traceability is independent of the kind of task for our experiment, but whether and how big a gain in correctness through traceability is, depends on the kind of task.

Table V
PERFORMANCE OF SUBJECTS PER TASK KIND

		Time [s]		Correct tasks	
Task	Trace	mean (sd)	diff	[%]	diff
Gantt A	no	1007 (416)	2%	33	140%
	with	1025 (424)		80	
Gantt B	no	974 (414)	-21%	21	150%
	with	769 (379)		53	
Gantt C	no	1020 (410)	-22%	57	-22%
	with	793 (378)		44	
Gantt D	no	843 (448)	-53%	27	239%
	with	393 (154)		90	
iTrust A	no	781 (384)	-17%	59	35%
	with	647 (280)		80	
iTrust B	no	840 (482)	-14%	53	14%
	with	722 (413)		60	
iTrust C	no	932 (403)	-20%	60	6%
	with	745 (278)		64	
iTrust D	no	827 (324)	-20%	22	215%
	with	665 (222)		70	

C. Task Order

This subsection focuses on the effect that the order in which a task was presented to a subject had on the performance of that subject for the particular task. The statistical tests show that the task order has a highly significant effect on a subjects' overall performance (see row "task order" in Table III: $Wilks'\lambda = 0.83, F = 9.02, ***$) as well as separately and even stronger on the time spent to complete a task, but not on the achieved correctness. That means that the first task solved by a subject was not less correct than the remaining ones, but considerably slower.

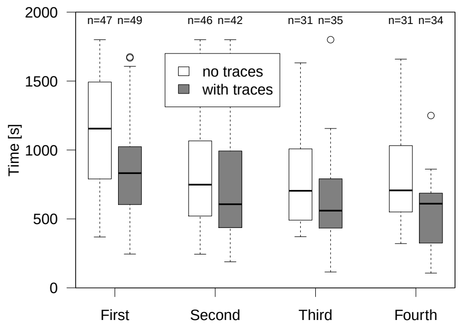

Figure 5. Time for completing a task, depending on the sequential order in which it was presented to the subject

Figure 5 compares in box plots the times that participants spend on tasks with and without traceability, separated for the four different orders in which a task could be solved. We omitted a plot of correctness as it is not significantly impacted by the task order, but Table VI presents complete descriptive statistics aggregated by task order.

Table VI
PERFORMANCE OF SUBJECTS FOR TASKS, GROUPED ACCORDING TO THE ORDER IN WHICH A TASK WAS PRESENTED TO THE SUBJECT

Order	Trace	Time [s] mean (sd)	Time [s] diff	Correct tasks [%]	Correct tasks diff
First	no	1134 (412)	-23%	30	140%
	with	877 (368)		71	
Second	no	808 (376)	-10%	41	56%
	with	730 (393)		64	
Third	no	786 (370)	-20%	48	48%
	with	631 (321)		71	
Fourth	no	823 (376)	-32%	58	11%
	with	559 (242)		65	

Subjects spent considerably more time on the first task presented to them, on average $1134s$ without and $877s$ with traceability, compared to the second performed task, which lasted on average $808s$ without and $730s$ with traceability. That is the so-called carryover or learning effect we expected, but that effect is independent from the traceability treatment as the statistical test suggests and the time differences within the table show. Focusing on correctness of solutions, it is interesting that tasks with traceability were solved with almost equal quality 64% to 71% fully correct solutions, while a task performed without traceability seems to become more often fully correct the later it has been performed by the user, 30% first task to 58% fourth task. Nonetheless, the interaction between traceability and task order is not statistically significant and so are the differences we found. Furthermore, it is not clear what exactly the reason for such an effect would be, a growing experience with the project or the traceability used for previous tasks. The study of that effect remains a future exercise.

V. Threats to Validity

This section discusses what is considered to be the most important threats to the validity of the experiment.

A. External Validity

Our experiment shows results of subjects with a spread of experiences, but with overall little industrial experience (on average 1 year) and does accordingly not allow us to draw conclusions for more experienced developers. Regarding that issue an additional study with more experienced subjects is required and planned. Furthermore, both projects are implemented in JAVA and though not expected, effects might be different for other programming languages.

We tried to keep the other aspects of the experiment as realistic and redundant as possible, applying two systems, using four tasks per project and having different kinds of tasks (bug reports vs. feature request). Systems, tasks and traces have been used in the original state as recovered from the system's archives. Focusing on the tasks that we selected, our results show that all are neither easy nor unsolvable. Subjects created 37% to 69% correct solutions for the eight different tasks. That observation suggests a balanced selection of tasks. Also, the tasks varied in the impact traceability had (in some cases, tasks with traces where faster, in other cases they were better, in yet other cases both). This variation also suggests a good selection of tasks. Nonetheless, the tasks had in common that they were understandable without a deep knowledge of the system and our results are only generalizable for that kind of task. Nonetheless, we believe that this would be the type of task that professionals less familiar with a project would be exposed to because more complex tasks are likely given to professionals who are familiar with the system and who are also likely not to benefit as much from traceability.

For both systems, we selected from a pool of 8 possible tasks. These tasks were related to the tools functionality and were understandable without knowledge of the working tool. We also made sure that the tasks involved no tool-specific techniques or libraries. The selection of tasks may thus pose a threat to validity. However, the chosen tasks were of different complexity, involving in some cases fewer and in other more artifacts.

We provided only traces as relevant for a given task to separate the benefit of traceability from its cost (cost=creating, maintaining, and retrieving them). This is reasonable because the cost of traceability is affected by a range of factors (e.g., levels of automation) and is likely to change with further traceability research. For that same reason, we also provided high quality traces only (correctness) to demonstrate the best possible benefit of traceability. However, incorrect and/or incomplete traceability are a practical problem and further studies are needed to investigate their impact on trace benefits (e.g., possibly involving existing trace technologies [17], [18]).

B. Internal Validity

To decrease variability in knowledge across participants we provided an introductory tutorial. The written form of the material minimized the possible influence of the experiments on the results. We asked subjects after the introduction whether they had questions before starting the experiment and found that except for a few organizational questions they felt ready. This suggests that the introduction was sufficient.

The time that subjects could spend on the experiment was restricted in two ways, raising the question whether the permitted time was sufficient or had an effect on the results. First, subjects had to stop working on a task after 30 minutes. This allowed time was more than twice as long as subjects on average spent on a task. Only 9 out of 315 tasks were stopped by the user or got terminated by the tool after 30 minutes. These observations suggest that the deadline per task had no relevant effect on our results. Second, subjects were permitted an overall time of two hours for performing tasks in order to ensure their consistent concentration. During that time subjects performed a minimum of 4 tasks, an average of 6.1 tasks, and 11 out of 52 subjects completed all 8 tasks. We analyzed results per task and not per subject and had assigned tasks in random sequence to subjects, ensuring their comparability. Furthermore, we studied the effect that the task order had on performance and found that after an initial learning effect between the first and the second performed task, the order does not significantly affect the performance of subjects.

Treatments were randomly assigned to the participants in order to balance learning effects. None of the participants knew the development perspective of the projects prior to the experiment. Learning effects was considered as independent variable and studied in depth.

We have used two different projects, with four different tasks each and we had a number of 315 performed tasks. Based on this diversity in the original experiment, we expect that replications of the experiment will offer results similar to those presented here. Concrete measured results will differ as they are specific to the subjects, but the underlying trends and implications should remain unchanged.

Our experiment aimed at evaluating the effect of traceability on software maintenance tasks. We decided to assess that effect by the performance of subjects implementing maintenance tasks and to operationalize performance by time to and correctness of solution. If a subject performs better on a maintenance task, then the time spent on the task should be lower and/or the solution should be more correct. Our experiment therefore focused on these measures.

VI. Cost-Benefit Estimation

Now that we assessed the benefit that requirements traceability can offer for performed maintenance tasks, the question arises of how that benefit compares to the overall costs of providing traceability for a development project. With this additional information it is possible to make a decision of whether traceability justifies its costs for a specific project or not. There are two costs drivers in having up-to-date traceability for a project: (1) the initial creation of traces, and (2) the update of traces after structural changes to traced artifacts. Unfortunately, no generic model for estimating the full costs of requirements traceability on different projects is available.

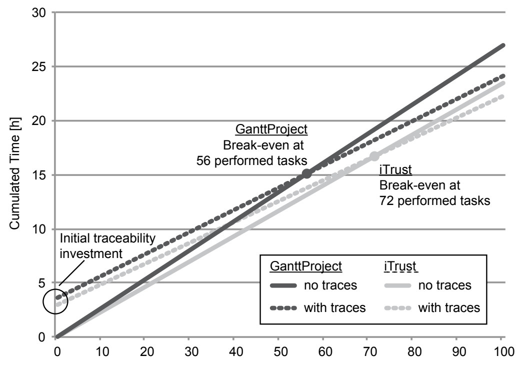

Figure 6. Cumulated time of 100 average tasks with and without traceability on GanttProject and iTrust. In terms of effort, the break-even point is reached with 56-72 tasks. The benefit of quality is not considered.

However, we can estimate the costs for the initial creation of traceability in both projects. In previous work, we measured and studied the creation of traceability links for GanttProject by its original developers [4]. This information does not only tell us how long the creation of all Gantt traces took (3.6 hours), but it also gives an estimation of how long a developer typically needs to trace methods in relation to their size (measured in LoC). Based on that information and based on additional experiments with students also tracing the Gantt source code, we previously proposed a simple model for estimating the costs of tracing Java methods in relation to their size. We estimated the costs for the creation of all iTrust traces based on that model as being 3 hours. Earlier analyses had showed that predicted values were within 7% of the actual costs. We compared the costs for setting-up traceability in GanttProject and iTrust with the costs of performing maintenance tasks with and without traceability. Figure 6 shows the incremental required time for performing 1 to 100 maintenance tasks with and without traceability on GanttProject and iTrust (the incremental time is an average across all tasks). We found that for Gantt after 56 performed tasks and for iTrust after 72 performed tasks, the initial costs for setting-up traceability would be justified and with each additional task the investment into traceability would payoff. These estimations can help decision makers that want to know whether traceability makes sense in their specific project. If the number of performed maintenance tasks is likely to supersede the discovered break-even points, then a benefit by introducing traceability is likely.

However, our estimation is imprecise in at least three regards. First, it does not take in consideration the cost for maintaining traces as we have no information regarding them. Additional studies are required on that topic. Second, we only consider the time saving of performed tasks with traces over those performed without traces for our estimation. But, subjects with traces also created considerably better solutions (60% more correct solutions). This quality difference will likely lead to a larger time gain in favor of traceability as tasks have to be fully solved either in additional tries by the same subject or by a more experienced developer. Third, existing traces may be used to support manifold additional development tasks with additional benefits. It is a future exercise to study the effect of traceability in support of other development activities.

VII. Conclusions

We conducted a controlled experiment with 52 subjects performing 315 real maintenance tasks on two third-party development projects: half of them with and the other half without traceability. Our finding is that subjects with traceability performed on average 21% faster on a task and created on average 60% more correct solutions. The effect of traceability is correlated with the kind of task. Especially, the gain in correctness varied quite strong among the tasks. Furthermore, our data indicates that the gain of correctness through traceability is shrinking after the first performed task. Subjects without traceability create more correct solutions after the initial task, while subjects with traceability perform very well right from the beginning. Finally, we aimed for an initial cost-benefit estimation and set the measured time reductions by using traceability in relation to the initial costs for setting-up traceability in the evaluated systems. Nonetheless, a proper cost-benefit model for traceability needs additional studies regarding the costs of maintaining traceability and regarding the benefit of traceability for other tasks in the software development process. This will be the focus of our future work. We are also planning to replicate the reported experiment with a group of experienced software developers in oder to study the influence of industry experience and trace quality.

Acknowledgments

We would like to thank all experiment participants and the developers of Gantt and iTrust for making their work publicly available. This work was funded by the Austrian Science Fund (FWF) M1268-N23 and P23115-N23, and the German Research Foundation (DFG) Ph49/8-1.

References

[1] P. Mäder, O. Gotel, and I. Philippow, “Motivation Matters in the Traceability Trenches,” in *Proc. 17th Int’l RE Conf. (RE’09)*, Atlanta, Georgia, USA, August 2009.

[2] M. Lindvall and K. Sandahl, “Practical implications of traceability,” *SPE*, vol. 26, no. 10, pp. 1161–1180, 1996.

[3] K. Pohl, *Process-Centered Requirements Engineering*. Research Studies Press, 1996, ISBN 0-86380-193-5.

[4] A. Egyed, F. Graf, and P. Grunbacher, “Effort and quality of recovering requirements-to-code traces: Two exploratory experiments,” in *18th IEEE Int’l RE Conf. (RE10)*, 2010, pp. 221–230.

[5] P. Arkley and S. Riddle, “Overcoming the traceability benefit problem,” in *13th Int’l Req. Eng. Conf.*, 2005, pp. 385–389.

[6] O. Gotel and A. Finkelstein, “An analysis of the requirements traceability problem,” in *Proc. 1st Int’l Conf. Req. Eng. ICRE94.*, Colorado Springs, CO, April 1994, pp. 94–101.

[7] B. Ramesh and M. Jarke, “Toward reference models of requirements traceability,” *IEEE TSE*, vol. 27, no. 1, pp. 58–93, 2001.

[8] A. Ahmad and M. Ghazali, “Documenting requirements traceability information for small projects,” *IEEE Int’l Multitopic Conference INMIC 2007.*, pp. 1–5, Dec. 2007.

[9] A. D. Lucia, R. Oliveto, and G. Tortora, “IR-based traceability recovery processes: An empirical comparison of ”one-shot” and incremental processes,” in *ASE*. IEEE, 2008, pp. 39–48.

[10] D. Cuddeback, A. Dekhtyar, and J. Hayes, “Automated requirements traceability: The study of human analysts,” in *18th IEEE Int’l RE Conf. (RE10)*, oct 2010, pp. 231 –240.

[11] P. Mäder and J. Cleland-Huang, “A visual traceability modeling language,” in *Model Driven Engineering Languages and Systems*, ser. LNCS. Springer, 2010, vol. 6394, pp. 226–240.

[12] R. L. Glass, *Facts and Fallacies of Software Engineering*. Boston, MA: Addison-Wesley Professional, 2002.

[13] W. J. Dzidek, E. Arisholm, and L. C. Briand, “A realistic empirical evaluation of the costs and benefits of UML in software maintenance,” *IEEE TSE*, vol. 34, no. 3, pp. 407–432, 2008.

[14] B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love, “Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics,” *IEEE TSE*, vol. 5, no. 2, pp. 96–104, 1979.

[15] (GanttProject). [Online]. Available: www.ganttproject.biz

[16] (iTrust). [Online]. Available: http://agile.csc.ncsu.edu/iTrust/

[17] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, “Fine-grained management of software artefacts: the ADAMS system,” *SPE*, vol. 40, no. 11, pp. 1007–1034, 2010.

[18] S. Klock, M. Gethers, B. Dit, and D. Poshyvanyk, “Traceclipse: an eclipse plug-in for traceability link recovery and management,” in *6th Int’l TEFSE Workshop*, 2011, pp. 24–30.