Empir Software Eng
DOI 10.1007/s10664-014-9314-z

Do developers benefit from requirements traceability when evolving and maintaining a software system?

Patrick Mäder · Alexander Egyed

Abstract Software traceability is a required component of many software development processes. Advocates of requirements traceability 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 71 subjects re-performing real maintenance tasks on two third-party development projects: half of the tasks with and the other half without traceability. Subjects sketched their task solutions on paper to focus on the their ability to solving the problems rather than their programming skills. Our findings show that subjects with traceability performed on average 24 % faster on a given task and created on average 50 % more correct solutions—suggesting that traceability not only saves effort but can profoundly improve software maintenance quality.

Keywords Requirements traceability · Software traceability · Software maintenance · Software evolution · Traceability usage · Traceability effect · Traceability benefit · Empirical software engineering · Controlled experiment · Study

Communicated by: Massimiliano Di Penta and Jonathan Maletic

Electronic supplementary material The online version of this article (doi:10.1007/s10664-014-9314-z) contains supplementary material, which is available to authorized users.

P. Mäder (✉)
Software Systems/Process Informatics Group, Technische Universität Ilmenau, Ilmenau, Germany
e-mail: patrick.maeder@tu-ilmenau.de

A. Egyed
Institute for Software Systems Engineering (ISSE), Johannes Kepler University, Linz, Austria
e-mail: alexander.egyed@jku.at

Published online: 22 June 2014

1 Introduction

Requirements-to-code traceability reflects the knowledge where requirements are implemented in the code. Its capture and maintenance is the focus of extensive research (Gotel et al. 2012; Cleland-Huang et al. 2014). Despite its growing popularity (Mäder et al. 2009; Rempel et al. 2013) surprisingly little is known about its benefits. Intuitively, requirements-to-code traces should be useful for many areas of software engineering. Researchers refer to better program comprehension and support for software maintenance (e.g., Lindvall and Sandahl 1996; Pohl 1996b). In the safety critical domain, traceability supports the certification process and is thus mandated by several standards (e.g., DO-178C (RTCA/EUROCAE 2011) and ISO 26262 (ISO 2011)). Nonetheless, there exists no empirical work in which the effect of requirements traceability was measured. Does it save effort? Does it improve quality?

The goal of this work is 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. Since developers tend to be less/not familiar with the code during maintenance, traces are perceived to be needed and useful. To study that effect, we carried out a large-scale study involving 71 subjects—from practitioners to students with a wide range of experiences. The subjects were asked to solve maintenance tasks taken from two software development projects: the open source Gantt Project (47 KLOC) and the iTrust system (15 KLOC). Eight tasks were selected, covering real bug fixes and feature extensions taken from the projects' documented archives. Tasks were randomly assigned to subjects, half the tasks with and the other half without traceability. Task solutions were recorded on paper and not implemented by the subjects. We measured the performance of subjects as the time they spent to solve a task and the correctness of their solution. The selected, real maintenance tasks also provided us with a gold standard as to how the original developers solved the given tasks. This allowed us to assess the correctness of the solutions the subjects provided for the tasks.

A secondary 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 varied based on subject experience, the kind of tasks the subjects were expected to solve, the different project domains, and others. All subjects had in common that they were not familiar with the projects we used in the study—a situation that commonly occurs during software maintenance and a situation under which developers are expected to benefit most from traceability.

In total, the subjects solved 461 tasks (i.e., 6.5 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 24 % faster on tasks and created on average 50 % 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. There are likely many subsequent benefits such as more effective maintenance, faster time to market, or less code degradation. We also found that some tasks benefited more from traceability than others, especially with regard to the correctness of the solution. Furthermore, we found that our observations were consistent regardless of subject experience and the project domain.

The implications of this study are numerous. Traceability strongly benefits software maintenance regardless of subject experience. While the capture and maintenance of traceability (studied in other works, e.g., Mäder et al. 2008; Egyed et al. 2010; Mäder and Gotel 2012) does require significant effort and while developers tend to perceive this activity to

be tedious and ineffective (Arkley and Riddle 2005), 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 also be the focus of our future work.

This paper extends our prior work presented at the 2012 International Conference on Software Maintenance (ICSM 2012) (Mäder and Egyed 2012) in several regards: (1) we added results from a group of 19 professional software developers replicating the experiment, (2) we discussed the effect of traceability on software maintenance in relation to the experience of subjects, (3) we analyzed the workflow and means of navigation applied by subjects, and (4) we added more detailed information about the performed tasks and the projects they were performed on.

The remainder of this paper is structured as follows. Section 2 discusses related studies in the area of requirements traceability and software engineering. Section 3 introduces our experimental set-up in depth, while Section 4 extensively reports about the results and findings of the experiment. In Section 5 we discuss the limitations of our study. Finally, Section 6 concludes and discusses possible areas of future investigations.

2 Related Work

Gotel and Finkelstein (1994) report the findings of a year-long empirical study into traceability conducted in 1992. The study involved around one hundred software development practitioners, holding a variety of positions within a large organization and 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 (2001) report on a large practitioner study of traceability where the data collection took over three years during the 1990's. The authors conducted a pilot study with fifty-eight master students in information technology 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 upon 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.

De Lucia et al. (2006) developed an Eclipse plug-in, called COCONUT (COde COmprehension Nurturant Using Traceability), which uses Latent Semantic Indexing (LSI) to indicate the similarity between source code under development and a set of high-level artifacts (e.g., requirements statements, use cases). The authors conducted a controlled experiment to assess the usefulness of the proposed approach. Master students were asked to perform development tasks with and without the use of COCONUT. Findings show that the tool significantly helps to improve the similarity between code and related requirements in presence of comments within the source code.

Arkley and Riddle (2005) report on a survey of nine software projects, small to multinational in scope, undertaken using questionnaires and interviews. 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. Several studies investigated the negative impact of inadequate traceability on software development. Researchers found that wrong granularity can lead to over-complex or inadequate traceability graphs, and thereby leads to project over-runs or software failures (Mäder et al. 2009; Leffingwell 1997). Ramesh et al. emphasized on the high costs of creating and maintaining traceability, which can only be compensated by higher quality and reduced overall costs if traceability is applied purposefully (Ramesh et al. 1995). Capturing software development activities to automatically generate traces may reduce trace capturing effort (Omoronyia and Sindre 2011). Though, neglecting traceability completely or capturing traces in an unstructured manner to reduce costs will lead to reduced system quality, expensive iterations of defect corrections, and increased project costs (Pohl 1996a; Dömges and Pohl 1998). Ahmad and Ghazali (2007) interviewed fifteen practitioners and analyzed project documentation of three IT companies. Their subjects had 6–10 years of practical experience in developing small projects. Their finding was that the interviewed subjects perceived pre-requirements traceability to be more beneficial than post-requirements traceability. Nonetheless, the authors did not study the benefits of traceability to their subjects in detail. In a recent publication (Mäder et al. 2013), we reported on traceability problems that were observed by members of the U.S. Food and Drug Administration (FDA) who systematically evaluated traceability documentation for FDA medical device approval. In this and other studies, it turned out to be a major problem that the definition of project-specific traceability strategies was either lacking or inappropriate for the investigated project (Mäder et al. 2013; Rempel et al. 2013, 2014). Briand et al. (2014) performed an extensive study evaluating whether and how design slices traced to safety requirements can support the safety inspection process. Similarly, a variety of other publications evaluated the effect of different traceability techniques in extensive studies, e.g., De Lucia et al. (2008); Cuddeback et al. (2010); Mäder and Cleland-Huang (2010, 2013); Mäder and Gotel (2012), but none of them evaluated the effect of traceability itself.

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 (Glass 2002) and traceability could reduce that cost. Accordingly, we decided to restrict our focus to software maintenance. We found several studies focusing on software maintenance, e.g., Dzidek et al. (2008) study the costs and benefits of UML documentation for software maintenance and Curtis et al. (1979) compare the performance of subjects solving maintenance tasks with complexity measures (e.g., Halstead and McCabe metrics) evaluating the same tasks. Furthermore, Ricca et al. (2010) performed a series of four experiments in different locations and with subjects possessing different experience, i.e., undergraduate students, graduate students, and research associates. The experiments aimed at comparing performances of subjects in comprehension tasks. The subjects were provided either with the source code complemented by standard UML diagrams or with the source code and diagrams stereotyped using the Conallen notation. Similarly to our experiment, the authors considered the experience of subjects and its influence on the treatment.

3 Method

A controlled experiment was conducted in which participants performed up to eight maintenance tasks with and without traceability. The research investigated the effort and quality differences of participants and assessed whether traceability significantly impacted them. The data was captured through six distinct lab sessions due to space and time restrictions. We analyzed all data together and therefore consider our study a single controlled experiment rather than a family of experiments (Basili et al. 1999). All factors considered relevant were either kept constant or treated as independent variables across the sessions. That means in particular that all subjects performed the same experiment in the same environment, instructions to subjects were provided in written form, and each session followed a predefined schedule. Furthermore, subjects were not allowed to interact with each other.

3.1 Participants

Our experiment involved 71 participants, 52 master-level computer science students of the JKU Linz and 19 professionals working as developers at software companies in and around Linz. Students were offered to credit their participation in our experiment, independently of their performance or any other factor, as a term obligation. We offered practitioners a fixed 100€ compensation for travel expenses also independently of their performance or any other factor. 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 7.1 years in software development and an average experience of 4.2 years in development with JAVA. On average, each subject had developed software in industrial environments for 2.8 years, though some had no industrial experience at all (30 students) and others up to 20 years. Based on that experience data, we decided that a categorization of participants into students and practitioners would not be a good discrimination for our data and rather analyzed results discriminated by development experience of subjects. None of the participants had previous knowledge of the source code used for the experiment.

3.2 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 gain project experience with every task (learning effect), we considered the order in which a task appeared. Finally, we expected subjects' development experience to possibly be related to the effect that traceability has and we decided to also consider it. 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 break down the work structure of a project into tasks, dependencies among tasks, and milestones

Table 1 Descriptive statistics and metrics of the software projects

Source metric	Gantt	iTrust
Number of files in project folder	1230	1651
Total lines of code (TLOC)	47k	15.5k
Number of classes (NOC)	626	232
Number of methods (NOM)	4570	1612
Number of packages (NOP)	51	16
Avg. method lines of code (MLOC)	5.51	5.38
Avg. depth of inheritance tree (DIT)	2.10	1.27
Max. depth of inheritance tree (DIT)	8	4
Cyclomatic complexity (VG)	1.70	1.55
Instability (RMI)[a]	0.44	0.31

[a]RMI = CE / (CA + CE), where CA (afferent coupling) measures the number of classes outside a package that depend on classes inside the package and CE (efferent coupling) measures the number of classes inside a package that depend on classes outside the package.

(Barashev and Thomas 2013). All the functionality of the tool deals with these charts. We selected GanttProject for our experiment, because it represents a class of development projects where documentation is kept to a minimum and traceability is perceived to be more beneficial if available. One of the key developers of the GanttProject provided the necessary requirements traces. The project can be characterized as:

- Real application, available for many years and widely used;
- Distributed, open-source development with multiple, diverse developers;
- Issue tracking system and source code repository keep track of changes and allowed for a detailed determination of bug reports and resulting code changes.

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 Williams et al. (2013) for several years. Though not an industrial project, iTrust is a complete application developed and improved in 15 major revisions. iTrust has been developed according to a regulatory code that, among other things, mandates rigorous documentation. Given this strong emphasis on documentation, we expected traceability to be less beneficial. In particular, we selected iTrust for the following reasons:

- Well documented, including substantial use cases, test cases and traceability;
- Source code, in accordance to own design notes and HIPPA[1] regulations;
- Use cases and source code versioned, allow determining associated changes.

Table 1 shows metrics of the two projects computed with the Eclipse Metrics plugin (2013). The metrics in the upper half of the table show that both projects are industrial size

[1]The U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA) contains a privacy rule, which regulates the use and disclosure of Protected Health Information (PHI). PHI is any information held by a covered entity, which concerns health status, provision of health care, or payment for health care that can be linked to an individual (HIPPA 1996).

and that Gantt is roughly three times larger than iTrust. We selected four additional metrics in order to measure the maintainability and complexity of both systems' source code. The average method lines of code metric (MLOC) captures the average length of a method across all methods of a system. Lengthy methods are considered harder to comprehend and to maintain. The depth of inheritance tree metric (DIT) counts the number of base classes for a class or a structure. Larger DIT values indicate more complex and harder maintainable code. The cyclomatic complexity metric (VG) aggregates across all methods of a system the number of decisions their code contains. Methods with a VG value above 15 are commonly considered hard to understand and hard to maintain. Finally, the instability metric (RMI) is an indicator of a package's resilience to change, ranging from 0 to 1. An RMI value of 0 indicates a stable package, while a value of 1 indicates an instable package. Instable packages are considered harder to maintain. All four metrics suggest well-structured and maintainable source code within both selected systems. However, the metrics also indicate that Gantt is slightly more difficult to maintain than iTrust, which correlates with our subjective opinion.

Task (Tk) altogether, the 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 2). For both systems, we selected from a pool of possible tasks so that the change or problem was related to the tool's functionality and was understandable without knowledge of the working tool. We also made sure that the tasks involved no tool-specific techniques or libraries. For Gantt we selected bug reports from the issue tracking system 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 2). iTrust tasks contained the original use case description and highlighted the required change or extension. While bug fixes and feature extensions do have different goals, the realization process to be followed by a developer not familiar with the source code is comparable. For both types of tasks, our participants had to understand the task to be solved, had then to explore the source code in order to identify the relevant part to be changed, and finally had to fill-in their solution into the questionnaire.

As an example, Fig. 1 shows the iTrust D task description as presented to the participants, the traces that were provided to them if the task was to be executed with traces, and a correct solution given by one subject. The task consists of the description of use case UC8, which had been expanded between Sep 7–16, 2009 with an additional sub-flow S2. Subjects were asked to describe necessary changes to the source code for implementing that change.

For each task, Table 2 also list the number of traces available for the traceability treatment, i.e., subjects performing the task with traces. For the tasks we were assigning to subjects, traceability was supposed to support stakeholders in identifying the source relevant for a given task, but not in implementing and testing their solution. For that reason, we decided that participants were not required to implement changes, but to identify the artifacts to be changed and to describe required changes in a structured questionnaire. Figure 1c shows a correct solution to the iTrust D task in the structured questionnaire given by one of the participants. By implementing the experiment that way, we focused our measurements on the relevant part of a task solution process and participants were able to solve more tasks in the given time span.

Traceability (Tr) Traces were either available to a subject on a particular task (with traces) or were not available (no traces). By trace we refer to an explicit connection between two

Table 2 Overview of all tasks and the number of provided traces per task

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

Full task descriptions can be found in the replication package and the project repositories

[a]Number of traces that were available to subjects performing the task with traces

development artifacts created by a subject matter expert. In our experiment, traces related requirements to the source code that was implementing them. Gantt tasks consisted of a bug description to be fixed. If the subject was asked to solve a Gantt task with traces, we provided traces available for the features impacted by the bug to be fixed. Traces were relating features to methods within the JAVA source code. Gantt traces were labeled in the experimenting tool with the feature name and the related JAVA method name (see left upper window on Fig. 2). iTrust tasks consisted of full use case descriptions with parts to be changed highlighted in the text (see Fig. 1a). On iTrust, traces related use cases scenarios to JAVA methods and to Java Server Pages within the source code (see Fig. 1b). If the subject was asked to solve an iTrust task with traces, all traces available for the use case to be

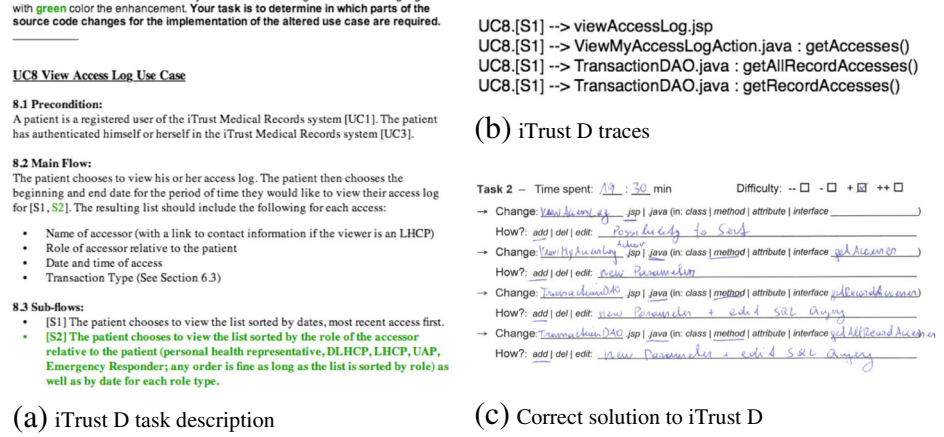


Fig. 1 Task description, provided traces, and sample of a correct solution given by one of the participants for iTrust D

changed were provided to the subject. Traces were labeled with the name of the use case scenario and the source code element they related. Traces were provided as they had been created by the original developers. We checked all traces for correctness and completeness

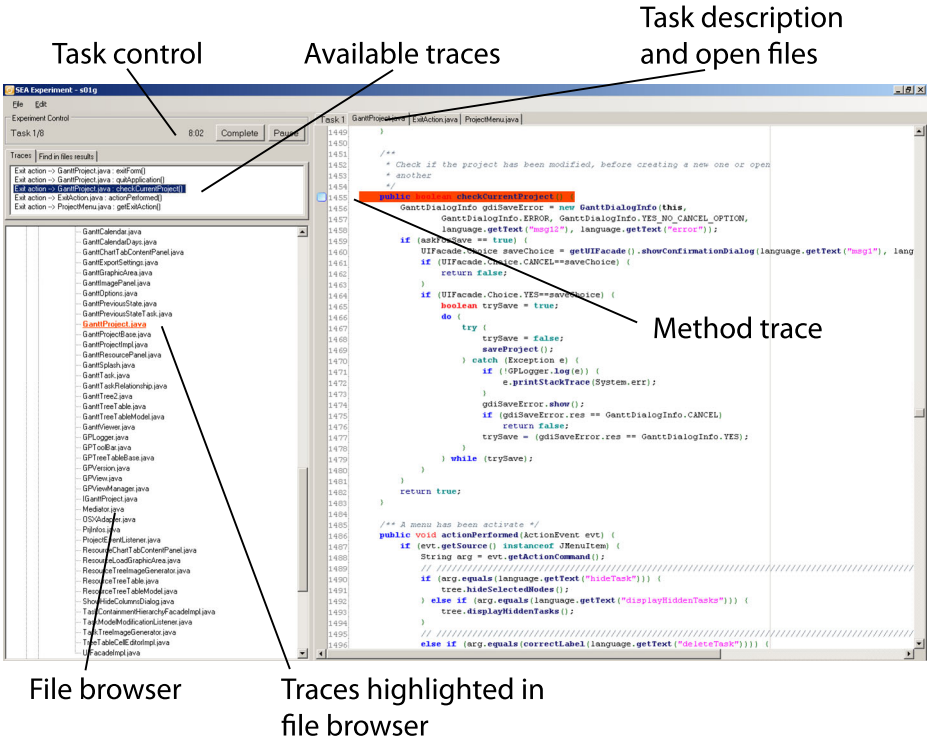


Fig. 2 The experimenting tool and its major features

and found no issues related to our tasks. The quality of traces should reflect the state-of-the practice for projects that chose to capture them. The number of available traces per task is shown in Table 2. The traces available to a subject did not merely restrict to the source code locations that had to be changed, meaning that they were not equal to the solution of a task and instead also traced to various other code locations implementing functionality of a use case or feature (traces and tasks are available as Electronic Supplementary Material to this paper). The only exception is the iTrust D task (see Fig. 1). iTrust D required a cross-cutting change to the system, spanning from the user interface to the data access layer and therefore impacted all source code implementing use case UC8. However, there was no hint for a subject indicating that all four code locations required changes.

Task Order (O) Tasks were assigned to subjects in different orders to avoid bias (assignment procedure discussed in Section 3.4). We thus captured the sequential order in which a task was presented to a subject as an independent variable. This variable is encoded as a factor with eight levels: Prj1/Tk1, Prj1/Tk2, . . . , Prj2/Tk4.

Development Experience (E) This variable captures the software development experience of a subject in general and is measured to the closest year. As students without industry experience can still have varying development experience, we asked each participant for both, her or his software development experience in general and her or his industrial development experience. On average, subjects had an extra 4.6 years of development experience that was not gained in industry. We decided to use the general development experience of subjects instead of their industry experience as an independent variable, because 30 students had no industry experience at all, but very diverging general experience. For experienced subjects, both measures are offset but correlated.

3.3 Dependent Variable

This study had one main 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 (see Section 3.5). We decided to measure the correctness of a solution as a factor with three levels: incorrect, partly correct, and fully correct. Since tasks were based on real changes, we knew the solution chosen by the original developers. We accepted alternative solutions, if they removed the bug or led to the desired result. For bugs, we considered a solution correct if it removed the problem, partly correct if the bug was localized but the provided solution would not or not entirely have removed it, and incorrect otherwise. All alternative solutions to remove a bug were localized within a few lines of code and therefore unproblematic to judge. For example, for the Gantt D task, where chart files with uppercase filename extension were not recognized by the Gantt tool (see Table 2), some subjects proposed to convert the file extension string to lowercase before comparison, while others proposed to ignore the case during comparison. Both solutions were accepted as correct. For feature extensions, we considered a solution correct if it realized the desired new functionality entirely, partly correct if the code to be extended was localized but the extension was not complete, and incorrect otherwise. Alternative solutions proposed by subjects were, for example, to include the functionality to be added directly into an existing method instead of creating and calling a new method as done by the original developers. We accepted both solutions as correct. We avoided controversial tasks and improved the task selection regarding that issue

through several pilot studies. The same grader scored all provided solutions. Thus, the performance measures of the experiment were the level of correctness that a solution had and the time required to provide that solution.

In addition to the performance of subjects as main dependent variable, we also captured subjects' workflows and their navigation strategies for solving a task as secondary measures. The workflow was operationalized as time that a subject spent for solving a task on three categories of documents: the description of a task (task description), source code that was irrelevant for solving a task (irrelevant source code), and source code that was relevant for solving a task (relevant source code). The navigation strategy was operationalized as how often each of the following four navigation types within the experimenting tool were applied by a subject: double click on an item in the project's file tree (tree), click on a trace in the project's trace list (trace), click on a retrieved file in the results list of the search functionality (search), and navigating through tabs of open files (tab). The aim of those two measures was understanding whether and how available traceability would impact the way in which a subject was solving a task.

Furthermore, we captured the difficulty in which a task was perceived by a subject as qualitative measure. The goal of this measure was to understand whether and how available traceability would impact how difficult a subject perceives a task and whether or not this perception correlates with the measured performance of a subject.

3.4 Experimental Design

In order to control for individual differences in performance, the experiment employed a $2 \times 2 \times 4 \times 4$ factorial design (Box et al. 2005). 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 every participant had 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.

3.5 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 using C#. We considered the advantages and disadvantages of utilizing a standard IDE (like Eclipse or Microsoft Visual Studio) and decided against it as we could not ensure that all subjects would be equally experienced in using the selected IDE, creating a possible bias in the results. Furthermore, a custom tool allowed us to fully control the experiment and to reduce the tool's functionality to what we considered relevant for evaluating traceability's effect for performing maintenance tasks, which we could not ensure for a standard IDE.

The developed tool was a text editor with development support (see Fig. 2). It provided a tree selector for browsing the project structure and opening files, multiple open files were supported through tabs, and the content of files was syntax highlighted (JAVA and JSP). Additionally, users could conveniently search either across all files of a project or within a selected file. Traces were provided in a separate window above the tree selector and were labeled with the names of the artifacts they were connecting (e.g., UC1[S1] $-$ > GetVisitReminders Action.java : getVisitReminders()). A click on a trace opened the related source

code file and highlighted the traced part. Furthermore, traces were highlighted in the file tree (see Fig. 2). The editor also controlled the experiment. Once, a task was started, its artifacts were displayed, i.e., the task description, the project file tree, and the captured traces, if the task was to be performed with traceability, were shown. As the tool controlled the experiment, we could capture exactly how much time a subject spent on a particular task. Furthermore, we captured how a subject explored a project in order to solve a task. That means we captured each artifact that a subject visited (source code artifact or task description), how she or he navigated to the artifact (through a trace, through the file browser, through a search, or by switching open tabs), and for how long the artifact was visited.

3.6 Procedure and Material

Material A packet of material was provided to 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 Barashev and Thomas 2013), 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 Williams et al. 2013). In all, the documentation for the iTrust system was more elaborate than the documentation for the GanttProject system. All material was provided in English. More details on tasks, the questionnaire, and the experimenting tool can be found in the Electronic Supplementary Material to this paper.

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 Section 3.1).

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. Relevant independent variables were controlled and each performed task was considered to be independent. Therefore, it was not mandatory that subjects were working on all tasks. The experimenting tool controlled the appearance of tasks (see Section 3.4 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. In pilot studies we found that subjects needed an average of 10–15 min to solve a task. Therefore, we considered a working period of up to 30 min on a task sufficient for the vast majority of participants (Section 5 will revisit that assumption). In order to avoid that subjects would solely focus on a single task throughout the whole experiment, the tool stopped their work after 30 min and required the next task to be started. For each task, the participants had to capture their changes in the questionnaire that provided templates for capturing changes (see Fig. 1c). Figure 1c shows a correct solution to task iTrust D. The participant performed iTrust D as her or his second task and captured along with all necessary changes to the source code, the time she/he spent on the task and the perceived difficulty. The manually captured time

was only meant as a backup and for cross-comparing the data captured by the tool. However, this backup was never needed. A check showed that the tool worked properly for all participants and so the time measured by the tool was used for analysis.

Debriefing At the end of the experiment we required each participant to fill in a short debriefing and feedback section of the questionnaire. That part captured whether or not the participant felt supported through traceability in solving the tasks and captured general feedback and suggestions regarding traceability and the experimenting tool.

4 Results

A multivariate analysis of variance (MANOVA) was conducted to determine the effect of the independent variables on the compound dependent variable performance. All statistics were performed with the R environment (R Project 2013). Multivariate normality of the captured data was tested using the multivariate version of the Shapiro-Wilk test ($W = 0.99$, $p - value = 0.08$). The test indicated a lack of evidence that the assumption of normality was violated. Since our data was normally distributed, a Bartlett test could be used to check homogeneity of variance within the continuous time data (*Bartlett's* $K^2 = 1.69$, $p - value = 0.19$). The test indicated a lack of evidence that the assumption of variance homogeneity was violated. Based on those 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. Following the principle of parsimony and the model simplification procedure proposed by Crawley (2002), which state that a simpler model should be preferred over a more complex model if fitting data equally well, the initial model was simplified by stepwise removal of non-significant interaction terms from the initial model. Eventually, we obtained the following simplified model:

$$y \sim Tr + P + Tk + O + E + Tr : Tk + Tr : O \tag{1}$$

The variable y refers to the dependent variable performance that is composed of time and correctness (see Section 3.3). The simplified model contains two interactions between factors, the first between traces and task ($Tr : Tk$) and the second between traces and task order ($Tr : O$). Table 3 presents the result of three model analyses, the multivariate MANOVA and the ongoing univariate ANOVAs, all computed for the simplified model. The univariate ANOVAs were computed in order to analyze, which individual influence a factor on the time and the correctness of a solution had.

From top to bottom, the rows of the table show statistical measures for the five independent variables as well as the two remaining interactions that 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

Table 3 Summary of multivariate and univariate analyses of variance ($n = 461$)

Source of variation	Df	Multivariate analysis of variance			Univariate analyses of variance			
		Performance			Time		Correctness	
		Wilks' λ	appr. F	Pr(>F)	F	Pr(>F)	F	Pr(>F)
Indep. variables								
Traces (Tr)	1	0.83	42.41	***	32.25	***	33.89	***
Project (P)	1	0.97	7.32	***	3.47	*	8.34	**
Task (Tk)	7	0.83	7.09	***	44.24	***	13.97	**
Task order (O)	7	0.76	9.05	***	81.61	***	8.53	n.s.
Experience (E)	3	0.90	7.62	***	3.40	n.s.	34.17	***
Interactions								
Tr x Tk	7	0.92	2.67	***	9.53	n.s.	18.82	**
Tr x O	7	0.95	1.74	*	3.43	*	5.84	n.s.

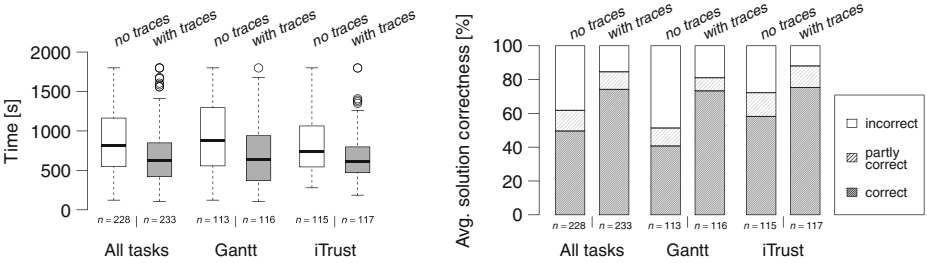
Significance codes for Pr(>F): > 0 '***'; > 0.001 '**'; > 0.01 '*'; > 0.025 'n.s.' (non significant)

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.

4.1 Traceability (Tr) and Project (P)

This subsection discusses the effects of the independent variables traceability and project on the performance of subjects. The statistical tests show that traceability significantly effects on how a subject overall performs for a particular task (see row "traces" of Table 3: $Wilks'\lambda = 0.83$, $F = 42.41$, $***$) as well as separately on the time she or he spent to complete the task and the correctness she or he achieved for the task. The project to which a task belongs also significantly affects the overall performance of subjects (see row "project" of Table 3: $Wilks'\lambda = 0.97$, $F = 7.32$, $***$), though the value of $Wilks'\lambda = 0.97$ shows that the project variable accounts for only 3 % of the variance in the results compared to 17 % of variance explained through the variable traces.

Figure 3a 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, Fig. 3b 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 incorrect solutions (white area) to a task in comparison to all solutions. The numbers below boxes and bars reflect the count of tasks aggregated in the particular group. For the same three groups (All tasks, Gantt tasks, iTrust tasks), Table 4 presents the mean and the standard deviation of time spent on performing a task (mean (sd)), the difference in mean values between both treatments in percentage of the treatment without traces (diff), the percentage of correct solutions that subjects created for particular tasks (Correct tasks), and the difference in correct replies between both treatments in percentage of the treatment



(a) Time spent on performing a task (b) Correctness distribution of results

Fig. 3 Performance of subjects with and without traceability across all performed tasks and per project

without traces (diff). The last column shows how difficult subjects perceived tasks per group on average. Perceived difficulty is presented as four level factor (very easy [−2, −1), easy [−1,0), hard [0,1), very hard [1,2]) and as computed mean value in brackets.

A comparison of all performed tasks shows that subjects working without traceability spent on average 889 *s* working on a task, while subjects working with traceability spent on average only 678 *s* working on a task. That means that on average traceability facilitated a 24 % faster task completion. Comparing the correctness of performed tasks, we found that participants working without traceability created 50 % correct solutions, while participants working with traceability created on average 74 % correct solutions. That means that traceability facilitates 50 % more correct solutions to a task. Furthermore, we found that subjects working with traceability perceived their task easier than those working without that support.

We compared time and correctness separately for Gantt and iTrust tasks. Regarding time, we found that subjects working with traceability spent on average slightly more time for completing a Gantt (692 *s*) vs. an iTrust task (665 *s*). For subjects working without traceability, we found a similar effect, but with a larger difference in times (Gantt (946 *s*) vs. iTrust (833 *s*)). Regarding correctness, subjects with traceability created 73 % (Gantt) and 75 % (iTrust) correct solutions for the tasks. The difference is considerably larger for subjects working without traceability, which created 41 % correct solutions for Gantt tasks vs. 58 % correct solutions to iTrust tasks. In summary, the differing project and task characteristics mainly effect subjects without traceability, while subjects with traceability perform almost equally (time and correctness) for both projects (compare also Fig. 3a and b).

Table 4 Performance of subjects across all tasks and per project

Project (P)	Traces	Time [s]		Correct tasks		Perceived
	(Tr)	Mean (sd)	diff	[%]	diff	difficulty
Both	No	889 (420)	−24 %	50	50 %	Hard (0.30)
	With	678 (347)		74		Easy (−0.33)
Gantt	No	946 (449)	−27 %	41	80 %	Hard (0.61)
	With	692 (400)		73		Easy (−0.14)
iTrust	No	833 (383)	−20 %	58	29 %	Easy (−0.01)
	With	665 (287)		75		Easy (−0.51)

4.2 Task (Tk)

The statistical tests show that the kind of task significantly affects a subjects overall performance (see row "task" in Table 3: $Wilks'\lambda = 0.83$, $F = 7.09, ***$) as well as separately the time spent on the task and the achieved correctness. Figure 4a compares the times that participants spent on tasks with and without traceability separately for the eight different tasks. Similarly, Fig. 4b compares the correctness that participants reached with and without traceability per type of task. Finally, Table 5 presents descriptive statistics.

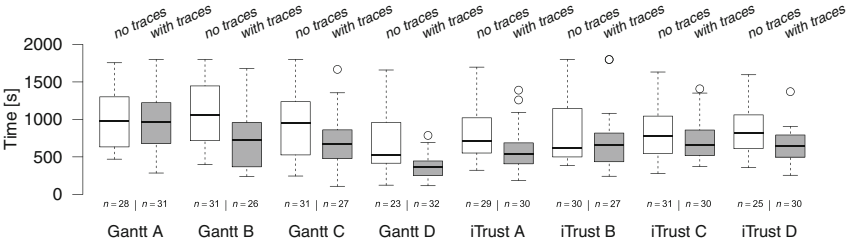
Especially, when comparing the results of different task types it is important to recognize that time and correctness are dependent. Someone who is solving a task correctly may require more time than someone who fails in solving a task. Regarding differences in time between tasks, subjects working with traceability completed their work, except for Gantt A, 15 % to 46 % faster than those performing the same tasks without traceability. For Gantt A they performed 1 % slower. The specific of the bug report used in Gantt A was that a multi-level, logical construct had to be understood in order to fix it. Subjects spent a long time on understanding that construct in order to create a correct solution. Especially, in comparison with correctness, showing 81 % vs. 39 % correct solutions created for Gantt A, we infer that subjects without traceability on average used less time, because many of them failed to identify the problem correctly.

Assessing correctness of solutions for the different kinds of task shows that subjects working with traceability created, except for iTrust B and C, 2 % to 188 % more fully correct solutions to tasks than those subjects working without traceability. For the iTrust B and C tasks, subjects created an almost equal number of fully correct solutions (0 % and −1 % difference). Though, for both tasks the combined percentage of partly and fully correct solutions is larger for the group working with traceability (see Fig. 4b) and subjects working with traceability were 17 % and 15 % faster respectively.

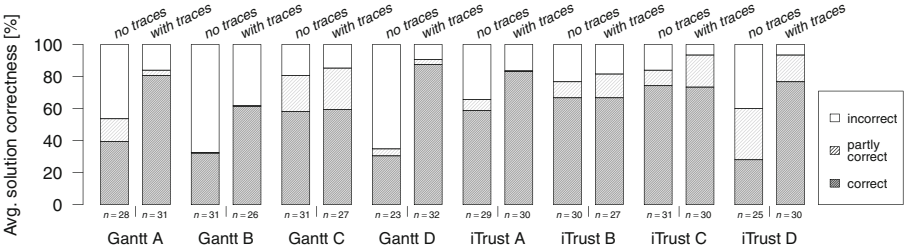
The results show that the level of achieved correctness is not only significantly dependent on the availability of traceability, but also on the type of task. Our statistical tests identified that interaction between traceability and kind of task to significantly affect the performance of subjects (see row "Tr x Tk" in Table 3: $Wilks'\lambda = 0.92$, $F = 2.67, ***$). The univariate analysis shows that the effect restricts to correctness only and does not significantly influence the time spent on a 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. That means that for some tasks traceability considerably improves correctness, while for other tasks it does not.

We analyzed the navigation strategy employed by subjects for solving a task. By navigation we refer to the way in which a subject did open files or navigated among open files through tabs. All subjects were able to navigate by three means: double clicking on a file name in the tree selector (tree), clicking on a file name retrieved by a search (search), and clicking on a tab containing an already opened file (tab). Additionally, for tasks solved with available traceability, subjects could click on a trace. Figure 4c shows a bar plot with the distribution of these four means of navigation in relation to all performed navigations per task. The analyzed and depicted values are mean percentages of navigation types referring to all solved tasks aggregated as bar. The number below each bar reflects the average number of navigations across all tasks aggregated in the bar.

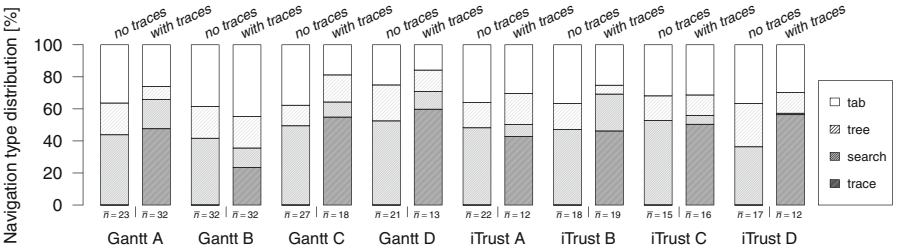
The plot shows that for tasks where traceability was available to subjects that traceability was used for 23–60 % of the navigations depending on the task. Traces were the favorite way of navigation for the subject having them available. Subjects working without traceability replace the missing trace navigation with searches (36–53 %). For tasks with traceability, the



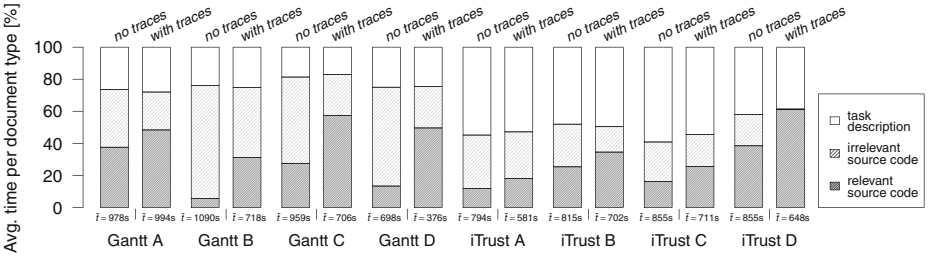
(a) Time for completing a task without and with traceability, depending on its type



(b) Correctness distribution of results created without and with available traceability, depending on the type of task



(c) Average navigation type usage depending on the kind of task



(d) Average time spent per document type depending on the kind of task

Fig. 4 Performance of subjects per task type

Table 5 Performance of subjects per type of task

Task (Tk)	Traces	Time [s]		Correctness [%]		Perceived
	(Tr)	Mean (sd)	diff	Correct	diff	difficulty
Gantt A	No	978 (379)	1 %	39	105 %	Very hard (1.04)
	With	984 (412)		81		Hard (0.69)
Gantt B	No	1090 (462)	−34 %	32	91 %	Very hard (1.14)
	With	718 (366)		62		Hard (0.00)
Gantt C	No	959 (452)	−26 %	58	2 %	Hard (0.50)
	With	706 (353)		59		Hard (0.11)
Gantt D	No	698 (433)	−46 %	30	188 %	Easy (−0.41)
	With	376 (164)		88		Very easy (−1.22)
iTrust A	No	794 (354)	−24 %	59	42 %	Easy (−0.36)
	With	603 (277)		83		Easy (−0.90)
iTrust B	No	846 (455)	−17 %	67	0 %	Hard (0.07)
	With	702 (377)		67		Easy (−0.54)
iTrust C	No	841 (387)	−15 %	74	−1 %	Hard (0.07)
	With	711 (266)		73		Easy (−0.40)
iTrust D	No	855 (333)	−24 %	28	174 %	Hard (0.20)
	With	648 (219)		77		Easy (−0.21)

search functionality has also been used, but to a much smaller extent (1–23 %). Navigation from the file tree has been used by subjects without traceability for 13–27 % and by subjects with traceability for 6–20 % of all navigations. The availability of traceability seems to be largely irrelevant for that type of navigation. Subjects without traceability applied tab navigation for 25–39 % of their navigations and subjects with traceability for 16–45 % of their navigations. Except for Gantt B, subjects with traceability used less tab navigation than those without traceability. In search for an explanation, we found that subjects regularly used traces also to focus files which were open already. That fact explains the lower number of tab navigations.

We also analyzed the amount of time that subjects spent on the three document categories: the task description, files that required a change in order to solve a task (relevant code), and files that were not required to be updated in order to solve a task (irrelevant code). Figure 4d shows the distribution of time spent on these types of document in relation to the overall time for solving a task. The number below each bar reflects the average time spent on a task across all tasks aggregated in the bar. The plot shows that across all tasks, subjects working with traceability spent more time (29 % Gantt A–458 % Gantt B) on relevant code than subjects working without traceability. Subjects working without traceability spent on most tasks considerably more time on irrelevant code. In relation to the performance of subjects, these results show that a faster and more correct task completion of subjects with traceability correlates with more time spent on relevant code.

Furthermore, the plot shows that the time spent on reading and understanding a task description is largely independent from whether subjects were working with or without traceability and also from the type of task, but it is clearly dependent on the project. For iTrust tasks, subjects spent almost half of their time on the task description. One explanation

is that Gantt tasks consisted of fewer text (bug descriptions filed to the issue tracking system), while iTrust tasks consisted of a whole and extensive use case description with some marked text indicating the change and subjects required more time to understand the task. Furthermore, subjects used the use case description of iTrust tasks for program comprehension. That explanation correlates with the observation that subjects working on iTrust tasks more frequently viewed the task description and that terms from the use case description were used for searches. These results also correlate with the lower gain in speed and correctness through traceability for iTrust compared to Gantt. The extensive amount of time spent on the task description seems to actually lower the effect of traceability.

4.3 Task Order (O)

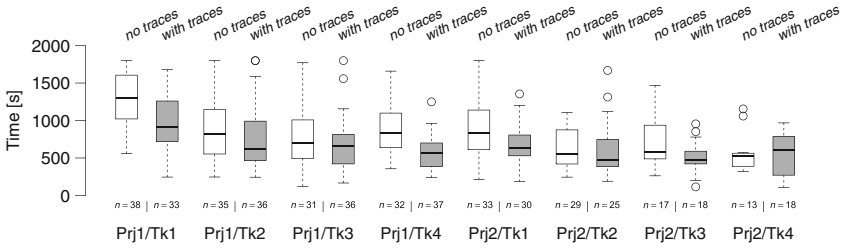
This subsection focuses on the effect that the order in which a task was presented to a subject had on the performance of that subject for the particular task. The statistical tests show that the task order significantly affects a subjects' overall performance (see row "task order" in Table 3: $Wilks'\lambda = 0.76, F = 9.05, ***$). Univariate analyses show that this effect restricts to the time spent on a task. The effect on correctness is not significant, meaning that the first task solved was considerably slower than the remaining ones.

Figure 5a and b compare the time spent on tasks and the correctness achieved, separated into the first to eight task solved by subjects (four tasks performed on the first project and four tasks performed on the second project). Table 6 presents descriptive statistics aggregated by task order.

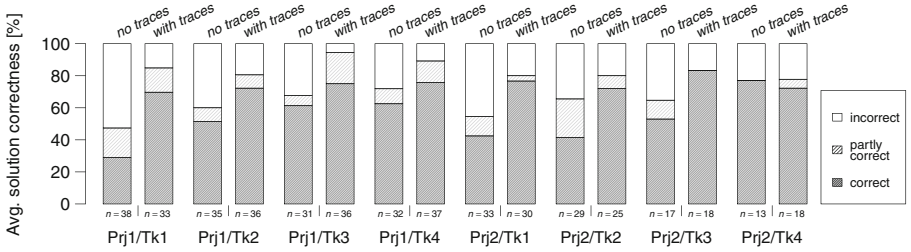
Subjects spent considerably more time on the first task per project, compared to the other performed tasks. That is the learning and familiarization effect we expected, but that effect is relatively independent from the traceability treatment as the statistical test suggests and the time differences within the table show. Focusing on correctness of solutions, it is interesting that tasks with traceability were solved with an almost constant quality of 70 %–83 % fully correct solutions, while a task performed without traceability seems to become more often fully correct the later it is performed in the task order. The statistical tests identified that interaction between traceability and task order to significantly effect the performance of subjects (see row "Tr x O" in Table 3: $Wilks'\lambda = 0.95, F = 1.74, *$). The univariate analysis shows that the effect restricts to time only and does not significantly influence the correctness achieved for a task. The finding is that the gain in speed through traceability is dependent on whether the task was performed as first or later task. The gain in correctness through traceability does not significantly dependent on whether a task was performed as first or later task by a subject.

Figure 5c shows the distribution of navigation types in relation to the order in which a subject solved it. Subjects working without traceability navigated on later tasks more often through searches than on the first performed task per project (except Prj2/Tk3). Accordingly, these subjects navigate on later tasks less often by tree and less often by tab. Subjects working with traceability apply on later tasks more trace and more search navigation (except Prj1/Tk3). Similar to the subjects working without traceability that difference results from less tree navigation and less tab navigation.

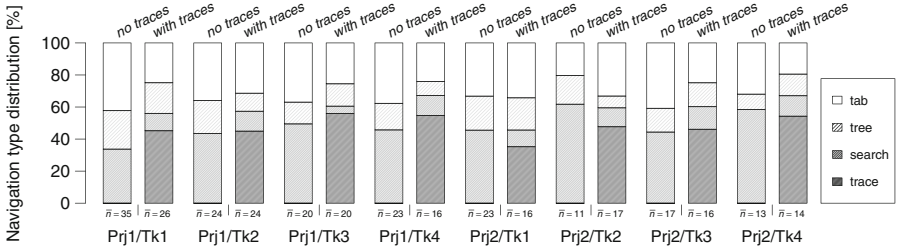
The data shows that subjects adopt the more sophisticated navigation functionality (trace and search) only slightly less on the first task. That finding is interesting for traceability as it demonstrates that subjects were using it right away from the first task they solved (implying a low learning curve). We also found that especially the first task per project, if performed without traceability, required more navigations than the remaining ones.



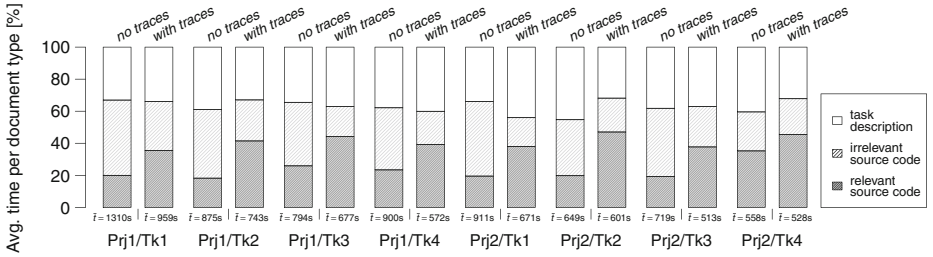
(a) Time to solution depending on the order in which a task was performed



(b) Correctness distribution of solutions depending on the order in which a task was performed



(c) Average navigation type usage depending on the order in which a task was performed



(d) Average time spent per document type depending on the order in which a task was performed

Fig. 5 Performance of subjects in relation to the order in which a task was presented

Figure 5d shows the effort distribution across document types in relation to the order in which a task was solved by a subject. The plot shows very small differences across

Table 6 Performance of subjects for tasks, grouped according to the order in which a task was presented to the subject

Task	Traces	Time [s]		Correct tasks		Perceived
order (O)	(Tr)	Mean (sd)	diff	[%]	diff	difficulty
Prj1/Tk1	No	1310 (368)	−26 %	29	141 %	Hard (0.94)
	With	968 (381)		70		Easy (−0.29)
Prj1/Tk2	No	900 (406)	−17 %	51	40 %	hard (0.12)
	With	743 (401)		72		Easy (−0.53)
Prj1/Tk3	No	794 (427)	−15 %	61	22 %	Easy (−0.1)
	With	677 (351)		75		Easy (−0.46)
Prj1/Tk4	No	897 (344)	−36 %	62	21 %	Hard (0.33)
	With	572 (234)		76		Easy (−0.33)
Prj2/Tk1	No	911 (381)	−26 %	42	81 %	Hard (0.79)
	With	670 (272)		77		Hard (0.03)
Prj2/Tk2	No	644 (270)	−7 %	41	74 %	Easy (−0.04)
	With	601 (347)		72		Hard (0.00)
Prj2/Tk3	No	719 (362)	−29 %	53	57 %	Hard (0.18)
	With	513 (209)		83		Easy (−0.61)
Prj2/Tk4	No	558 (259)	−5 %	77	−6 %	Easy (−0.73)
	With	528 (275)		72		Easy (−0.44)

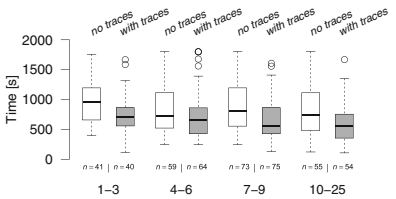
the performed tasks with and without traceability. Considering the trend that the time for performing a task becomes lower with each task a subject performs, the time distribution across the document types shows that subjects spent less time on all three kinds of document, as opposed to spending less time on irrelevant code as one might suspect.

4.4 Experience (E)

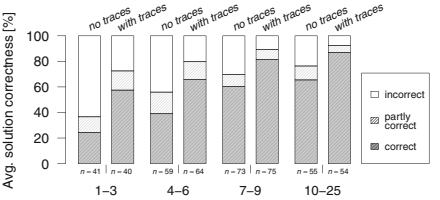
This subsection focuses on the effect that the experience of subjects had on their performance, possibly in interaction with traceability. The statistical tests show that experience significant affects a subjects' overall performance (see row "experience" in Table 3: $Wilks'\lambda = 0.90, F = 7.62, ***$) and also separately the correctness achieved for a task. That means that more experienced subjects solved tasks more correctly, independent of the availability of traceability.

The Fig. 6a and b compare time and correctness of performed tasks separated into four groups of development experience. We decided for those four groups while aiming for a comprehensible number of groups and aiming for an almost equally distributed number of subjects per group (14 students with 1–3 years of experience, 19 students/1 practitioner with 4–6 years of experience, 15 students/7 practitioners with 7–9 years of experience, and 4 students/11 practitioners with more than 9 years of experience). Table 7 presents descriptive statistics aggregated by the same four groups of experience.

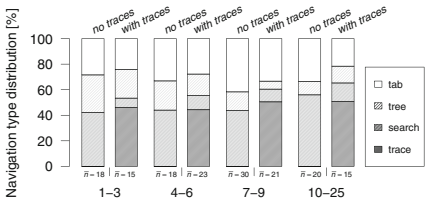
We can observe a trend showing that with growing experience the difference in correctness between tasks solved with and without traceability is shrinking, while the difference in



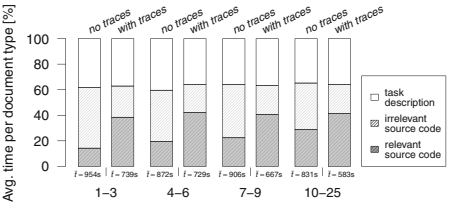
(a) Time for completing a task, depending on the development experience of the subject [years]



(b) Correctness distribution of solutions depending on the development experience of the subject [years]



(c) Average navigation type usage depending on the development experience of the subject [years]



(d) Average time spent per document type depending on the development experience of the subject [years]

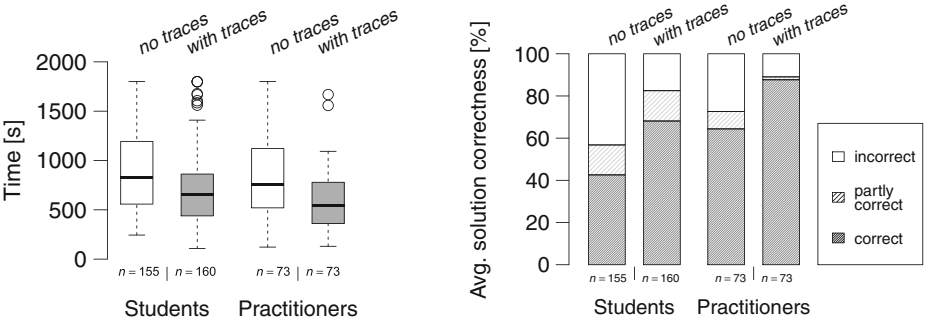
Fig. 6 Performance of subjects in relation to their development experience

time stays almost equal. That observation suggests that subjects with more experience create more correct solutions by default and therefore gain less support through traceability. However, independently of their experience all subjects benefited from traceability in terms of speed and correctness.

Figure 6c shows the distribution of navigation types in relation to the experience of the subjects that were performing the task. The figure shows a tendency, independent of available traceability, that more experienced subjects did use searches more frequently for navigating and so replace tree navigation, which is more frequently used by less experienced subjects. However, traceability navigation appears to largely replace search-based navigation when available and its frequency of use is almost independent of experience.

Table 7 Performance of subjects on tasks grouped according to their development experience

Experience	Traces	Time [s]		Correct tasks		Perceived
(E) [years]	(Tr)	Mean (sd)	diff	[%]	diff	difficulty
1–3	No	954 (357)	−23 %	24	136 %	Very hard (1.02)
	With	739 (328)		57		Hard (0.13)
4–6	No	879 (431)	−16 %	39	68 %	Hard (0.25)
	With	735 (413)		66		Easy (−0.16)
7–9	No	906 (441)	−26 %	60	35 %	Hard (0.03)
	With	667 (310)		81		Easy (−0.57)
10–25	No	834 (417)	−30 %	65	33 %	Hard (0.15)
	With	583 (309)		87		Easy (−0.52)



(a) Time for completing a task by students vs. practitioners

(b) Correctness distribution of solutions provided by students vs. practitioners

Fig. 7 Performance of students vs. practitioners

Figure 6d shows the effort distribution across document types in relation to the experience of the subject that was performing the task. For subjects working with traceability, the plot shows no significant differences. For subjects working without traceability, the plot shows that more experienced subjects spent more time on relevant code and less time on irrelevant code.

Figure 7 compares time and correctness of performed tasks separated for students and practitioners. We decided to categorize subjects based on their development experience rather than based on whether they are a student or a practitioner, because we found the experience to be more descriptive (see Section 3.1). However, for experiments with students usually the question arises how realistic their participation is. The figures shows that practitioners on average performed faster and more correct on a task than students, but both groups experienced an almost equal gain in speed and correctness. Suggesting again that the effect of traceability exists independently of a subject's experience.

4.5 Qualitative Feedback

In the debriefing section of the questionnaire, we asked participants whether they felt supported by traceability in performing the tasks on which it was available. All but two participants replied that they felt supported. Several subjects underlined the ticked "yes" box or put "very" behind it.

We also asked whether and how they would improve the provided traceability and if they had comments on the experiment. Half of the participants (32 out of 71) replied on that question. The replies fall into three categories: traceability, tasks, and tool.

Traceability In their comments, multiple participants positively referred to traceability in general, for example, "[traceability] really helped me", "with traces available, I found the code sufficiently documented and have no improvement suggestions.", "[traceability] put my focus on the relevant parts [of the code] and prevented me from wasting my time with understanding irrelevant code", and "traceability made solving the tasks much easier". One subject suggested an additional trace for Gantt. Two subjects referred to the fact that all traces for a use case (iTrust) or a feature (Gantt) were provided and not only the ones that were need to solve the task: "I had the feeling that not every trace was really needed to solve the task" and "traceability was not

always accurate enough". One subject found "iTrust traces more helpful than Gantt traces". Several participants requested additional documentation for the provided traces, for example, "Sometimes traces confused me. It would be nice to have comments for them.", "provide comments for traces", and "provide an overview and information on traced methods". Other improvement suggestions were: "trace more directly to the impacted code and not only to a method's corpus", "trace and highlight previous edits to the code", and "show hierarchy in the traces according to the call hierarchy of the traced methods".

Tasks Three replies referred specifically to the given tasks. Two of them referred to the difficulty of the tasks "the tasks were very complicated for me" and "I'm programming for 4 years, but I'm not experienced with such large systems." Similar to the replies that requested explanation and documentation on traces, on participant requested more comments in the code: "The traceability was helpful, but it is hard to understand a new system. More comments [in the code] would be helpful."

Tool Multiple participants suggested tool improvements, for example, "allow to search within the traced code only" and "provide a visual overview of documents and highlight relevant regions". Several participants requested more advanced IDE functionality like "show call hierarchy like in Eclipse" and "jump to declaration and references of a method and show an outline of the current class".

5 Threats to Validity

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

5.1 External Validity

Our experiment shows results of subjects with a wide spread of development experiences, allowing us to draw conclusions for that population. Both employed projects were 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 projects, using four tasks per project and having different kinds of tasks (bug reports vs. feature requests). Projects, tasks and traces have been used in the original state as taken from the projects' repositories. Focusing on the tasks that we selected, our results show that all are neither easy nor unsolvable. We had on average 28 % to 88 % correct solutions for the different tasks with and without traceability. They represent a type of tasks that is understandable without a deep knowledge of the project and our results are only generalizable for that kind of tasks. Nonetheless, we believe that this would be the type of tasks that professionals joining an unknown project would most likely be exposed to, creating an overall representative scenario for maintenance tasks with available traceability. 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.

5.2 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. None of the participants knew the development perspective of the projects prior to the experiment.

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 min. This allowed time was more than twice as long as subjects on average spent on a task. Only 9 out of 461 tasks were stopped by the user or got terminated by the tool after 30 min. 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 6.63 tasks, and 31 out of 71 subjects completed all 8 tasks. The completion rate is distributed across students as follows: 8 completed four tasks, 9 completed five tasks, 13 completed six tasks, 7 completed seven tasks, and 15 completed all eight tasks. Among the practitioners, 16 completed all eight tasks and one each completed five, six, and seven 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 did not significantly affect the performance of subjects.

5.3 Conclusion Validity

Possible threats to statistical conclusion validity include violations of the assumptions underlying statistical procedures, low statistical power, and low effect size. To reduce any impact of violations in the assumptions, our design of an experiment with randomized treatment assignment and with equal group size helped eliminate these impacts. Before making inferences from statistical test, the MANOVA assumptions were verified. Multivariate normality was tested and concluded using the Shapiro-Wilk test and a Bartlett test was used to check homogeneity of variance within the continuous time data.

5.4 Reliability

We used two different projects, with four different tasks each and we had 461 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.

5.5 Construct Validity

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 performed 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.

6 Conclusions

We conducted a controlled experiment with 71 subjects re-performing 461 real maintenance tasks on two third-party development projects: half of them with and the other half without traceability. Task solutions were sketched and recorded on paper to avoid that programming skills bias the results. Our finding is that traceability had a significant effect on the performance of subjects conducting maintenance tasks, measured as combined effect of both time to and correctness of solution.

We found that subjects with traceability performed on average 24 % faster on a task and created on average 50 % more correct solutions. These observations correlate with the perception of our subjects. All but two participants replied after the experiment that they felt supported by traceability in performing the tasks, often responding very enthusiastic towards traceability.

The effect of traceability is correlated with the kind of tasks. Especially, the gain in correctness varied quite strongly 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. We further found that development experience effects the performance of subjects, but does not significantly influence the effect of traceability. Traceability also facilitates a higher performance of subjects independent of their development experience.

In future work we will focus on analyzing the cost related to the creation and maintenance of traceability. We are also planning to replicate the reported experiment in order to study the influence of trace quality on the effect of traceability.

Acknowledgments We would like to thank all participants for their dedicated work 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, P23115-N23, the German Research Foundation (DFG) Ph49/8-1, and the German Ministry of Education and Research (BMBF) 16V0116.

References

Ahmad A, Ghazali M (2007) Documenting requirements traceability information for small projects. In: IEEE international multitopic conference (INMIC 2007), pp 1–5. doi:10.1109/INMIC.2007.4557711

Arkley P, Riddle S (2005) Overcoming the traceability benefit problem. In: Proceedings of the 13th international requirements engineering conference (RE05), pp 385–389. doi:10.1109/RE.2010.35

Barashev D, Thomas A (2013) GanttProject: Free project scheduling and management. http://www.ganttproject.biz

Basili V, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Trans Softw Eng 25(4):456–473. doi:10.1109/32.799939

Box GEP, Hunter JS, Hunter WG (2005) Statistics for experimenters: design, innovation, and discovery, 2nd edn. Wiley, ISBN 978-0-471-71813-0

Briand L, Falessi D, Nejati S, Sabetzadeh M, Yue T (2014) Traceability and sysml design slices to support safety inspections: a controlled experiment. ACM Trans Softw Eng Methodol (TOSEM) 23(1):1–43. doi:10.1145/2559978

Cleland-Huang J, Gotel O, Huffman Hayes J, Mäder P, Zisman A (2014) Software traceability: trends and future directions. In: Proceedings of the 36th international conference on software engineering (ICSE). Hyderabad. pp 55–69. doi:10.1145/2593882.2593891

Crawley MJ (2002) Statistical computing—an introduction to data analysis using S-plus. Wiley, ISBN 978-0-471-56040-1

Cuddeback D, Dekhtyar A, Huffman Hayes J (2010) Automated requirements traceability: the study of human analysts. In: Proceedings of the 18th IEEE international requirements engineering conference (RE10), pp 231–240. doi:10.1109/RE.2010.35

Curtis B, Sheppard SB, Milliman P, Borst MA, Love T (1979) Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. IEEE Trans Softw Eng 5(2):96–104. doi:10.1109/TSE.1979.234165

De Lucia A, Oliveto R, Zurolo F, Di Penta M (2006) Improving comprehensibility of source code via traceability information: a controlled experiment. In: Proceedings of the 14th international conference on program comprehension (ICPC 2006), pp 317–326. doi:10.1109/ICPC.2006.28

De Lucia A, Oliveto R, Tortora G (2008) IR-based traceability recovery processes: an empirical comparison of "one-shot" and incremental processes. In: Proceedings of the 23rd IEEE/ACM international conference on automated software engineering (ASE 2008). IEEE Computer Society, pp 39–48. doi:10.1109/ASE.2008.14

Dömges R, Pohl K (1998) Adapting traceability environments to project-specific needs. Commun ACM 41(12):54–62. doi:10.1145/290133.290149

Dzidek WJ, Arisholm E, Briand LC (2008) A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Trans Softw Eng 34(3):407–432. doi:10.1109/TSE.2008.15

Eclipse Metrics plugin (2013) Eclipse metrics plugin, http://sourceforge.net/projects/metrics/

Egyed A, Graf F, Grünbacher P (2010) Effort and quality of recovering requirements-to-code traces: two exploratory experiments. In: Proceedings of the 18th IEEE international requirements engineering conference (RE), pp 221–230. doi:10.1109/RE.2010.34

Glass RL (2002) Facts and fallacies of software engineering. Addison-Wesley Professional, Boston. ISBN 978-0321117427

Gotel O, Finkelstein A (1994) An analysis of the requirements traceability problem. In: Proceedings of the 1st international conference on requirements engineering ICRE94. Colorado Springs, pp 94–101. doi:10.1109/ICRE.1994.292398

Gotel O, Cleland-Huang J, Huffman Hayes J, Zisman A, Egyed A, Grünbacher P, Dekhtyar A, Antoniol G, Maletic J, Mäder P (2012) Traceability fundamentals. In: Cleland-Huang J, Gotel O, Zisman A (eds) Software and systems traceability. Springer, London, pp 3–22. doi:10.1007/978-1-4471-2239-5_1

HIPPA (1996) Health insurance portability and accountability act (HIPPA)—wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Health_Insurance_Portability_and_Accountability_Act&oldid=593856731 [Online; accessed 14-February-2014]

ISO (2011) ISO:26262-6:2011 Road vehicles—functional safety—part 6: product development at the software level

Leffingwell D (1997) Calculating your return on investment from more effective requirements management. Tech. rep., Available online at http://www.ibm.com/developerworks/rational/library/347.html

Lindvall M, Sandahl K (1996) Practical implications of traceability. Softw Pract Exp 26(10):1161–1180. doi:10.1002/(SICI)1097-024X(199610)26:10<1161::AID-SPE58>3.0.CO;2-X

Mäder P, Cleland-Huang J (2010) Model driven engineering languages and systems—13th international conference, MODELS 2010, Oslo, Norway, October 3–8, 2010, Proceedings, Springer, vol LNCS 6394, pp 226–240. doi:10.1007/978-3-642-16145-2_16

Mäder P, Cleland-Huang J (2013) A visual language for modeling and executing traceability queries. Softw Syst Model 12(3):537–553. doi:10.1007/s10270-012-0237-0

Mäder P, Egyed A (2012) Assessing the effect of requirements traceability for software maintenance. In: Proceedings of the 28th IEEE international conference on software maintenance (ICSM). IEEE Computer Society, pp 171–180. doi:10.1109/ICSM.2012.6405269

Mäder P, Gotel O (2012) Towards automated traceability maintenance. J Syst Softw 85(10):2205–2227. doi:10.1016/j.jss.2011.10.023

Mäder P, Gotel O, Philippow I (2008) Rule-based maintenance of post-requirements traceability relations. In: 16th IEEE international requirements engineering conference (RE'08), 8–12 September 2008, Barcelona, pp 23–32. doi:10.1109/RE.2008.24

Mäder P, Gotel O, Philippow I (2009) Motivation matters in the traceability trenches. In: Proceedings of the 17th international requirements engineering conference (RE09), Atlanta, pp 143–148. doi:10.1109/RE.2009.23

Mäder P, Jones PL, Zhang Y, Cleland-Huang J (2013) Strategic traceability for safety critical projects. IEEE Softw 30(3):58–66. doi:10.1109/MS.2013.60

Omoronyia I, Sindre G (2011) Exploring a bayesian and linear approach to requirements traceability. Information and Software Technology 53(8):851–871. doi:10.1016/j.infsof.2011.03.001

Pohl K (1996a) PRO-ART: enabling requirements pre-traceability. In: Proceedings of the 2nd international conference on requirements engineering (ICRE). IEEE Computer Society, pp 76–84. doi:10.1109/ICRE.1996.491432

Pohl K (1996b) Process-centered requirements engineering. Research Studies Press, ISBN 0-86380-193-5

R Project (2013) The R project for statistical computing. http://www.r-project.org

Ramesh B, Jarke M (2001) Toward reference models of requirements traceability. IEEE Trans Softw Eng 27(1):58–93. doi:10.1109/32.895989

Ramesh B, Powers T, Stubbs C, Edwards M (1995) Implementing requirements traceability: a case study. In: Proceedings of the 2nd IEEE international symposium on requirements engineering (ISRE), pp 89–95. doi:10.1109/ISRE.1995.512549

Rempel P, Mäder P, Kuschke T (2013) An empirical study on project-specific traceability strategies. In: Proceedings of the 21st international requirements engineering conference (RE13). Rio de Janeiro, pp 195–204. doi:10.1109/RE.2013.6636719

Rempel P, Mäder P, Kuschke T, Cleland-Huang J (2014) Mind the gap: assessing the conformance of software traceability to relevant guidelines. In: Proceedings of the 36th international conference on software engineering (ICSE14). Hyderabad, pp 943–954. doi:10.1145/2568225.2568290

Ricca F, Di Penta M, Torchiano M, Tonella P, Ceccato M (2010) How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes: a series of four experiments. IEEE Trans Softw Eng 36(1):96–118. doi:10.1109/TSE.2009.69

RTCA/EUROCAE (2011) DO-178C/ED-12C: software considerations in airborne systems and equipment certification

Williams L, Meneely A, Smith S, Hayward L, Smith B, King J (2013) iTrust: role-based healthcare. http://agile.csc.ncsu.edu/iTrust/

Patrick Mäder is a senior researcher at the Technische Universitt Ilmenau, Germany. His research interests include software engineering with a focus on requirements traceability, requirements engineering, and object-oriented analysis and design. Mäder received a Diploma degree in industrial engineering and a PhD degree (Distinction) in computer science from the Technische Universitt Ilmenau in 2003 and 2009, respectively. He worked as a consultant for the EXTESSY AG, Wolfsburg, as postdoctoral fellow at the Institute for Systems Engineering and Automation (SEA) of the Johannes Kepler University, Linz, and as postdoctoral researcher at the Software and Requirements Engineering Center at the DePaul University, Chicago.

Alexander Egyed received the doctorate degree from the University of Southern California (USC). He is a full professor at the Johannes Kepler University (JKU), Austria. He was with Teknowledge Corporation (2000–2007) and University College London, United Kingdom (2007–2008). He is most recognized for his work on software and systems modeling - particularly on consistency and traceability of models. His work has been published in more than 100 refereed scientific books, journals, conferences, and workshops, with more than 3,000 citations to date. He was recognized as the 10th best scholar in software engineering in Communications of the ACM, was named an IBM Research Faculty Fellow in recognition to his contributions to consistency checking, received a Recognition of Service Award from the ACM, a Best Paper Award from COMPSAC, and an Outstanding Achievement Award from USC. He has given many invited talks, including four keynotes, served on scientific panels and countless program committees, and has served as program (co)chair, steering committee member, and editorial board member. He is a member of the IEEE, IEEE Computer Society, ACM, and ACM SigSoft.