Do Software Engineers benefit from Source Code Navigation with Traceability? – An Experiment in Software Change Management

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

***Abstract*—For decades now, mainstream development environments provide the same basic automations for navigating source code: mainly searching and the tree exploration of files and folders. This may imply that other automations have little additional value or too steep a learning curve for mainstream adoption. This paper investigates whether existing navigation automations enriched with traceability benefit basic maintenance tasks such as changing features and fixing bugs in code. To test this, we conducted a controlled experiment with 52 subjects performing real maintenance tasks on two third-party development projects: all with the same navigation tool but half of the tasks with and the other half without traceability navigation. We found that the existence of traceability profoundly affected the quality of the change tasks and fundamentally changed how software engineers navigated through source code. We show that software engineers benefit instantly from traceability, without training, which is to show that the current automations available to software engineers are by no means sufficient or the only easy ones to use.**

I. Introduction

Traceability relations capture dependencies of artifacts created during the development of a software system. Advocates of traceability cite advantages like easier program comprehension and support for software maintenance. Despite a growing popularity, there is little published evaluation about the use of traceability and stakeholders still perceive the creation and maintenance of traceability links to be tedious and ineffective [1].

It is our assumption that a software engineer is expected to benefit from traceability much like she or he is expected to benefit from other automations available today: such as search capabilities or project folder exploration capabilities. To test that assumption, we set-up a study with 52 subjects performing real maintenance tasks on two third-party development projects. For half of the tasks, the subjects were also given a special automated navigation support based on traceability information whereas for the other half of the tasks this special capability was not provided. The navigation capability based on traceability showed where requirements affected by change were implemented.

The goal of this work is to investigate whether software engineers benefit from those capabilities. A change task involves three activities: 1) understanding the problem (change task), 2) searching for all relevant places in the source code where a change task needs to be realized, and 3) implementing the change (changing the code). Automations for navigation are not expected to support activities 1 and 3 in a significant manner. However, activity 2 should particularly benefit from navigation automations based on traceability. The goal of this work to study automations that are available to a software engineer for navigation and that use traces.

While to date no automation exists for creating and maintaining traceability between requirements and code, this paper provides the motivation that such automation would be highly beneficial. This motivation is based on the observation that subjects working with the traceability navigation were not only 21% faster and 60% more correct but used the navigation features available to them in a profoundly different manner. The data suggests that state-of-the-practice automations that support software engineering change tasks are by no means sufficient and navigation based on traceability holds a large potential in improving software engineering practice.

II. Related Work

Maintenance is a major cost driver in the development of a system and traceability navigation could potentially reduce that costs, but so far no empirical evidence is available. There are several studies focusing on software maintenance, e.g., Dzidek et al. [2] study the costs and benefits of UML documentation for software maintenance and Curtis et al. [3] compare the performance of subjects solving maintenance tasks with complexity measures (e.g., Halstead and McCabe metrics) evaluating the same tasks. Ko et al. [4] study ten developers while trying to understand unfamiliar code. The authors state that: "Eclipse's navigational tools caused significant overhead" during that process. Subjects collected relevant information as file tabs and others, but that markers got lost again as these interfaces were used for other tasks. On average, 35% of the time was spent on mechanics of navigation within and between files – quite consistent with an observation by Glass [5]. Empirical work on requirements traceability was focused on very general questions [6], [7], but not on assessing the actual effect of traceability for certain development activities.

III. Method

Participants: Subjects comprised 52 students of computer science, studying at the JKU Linz. These participants had an average experience of 5.3 years in software development and had on average spent one of these years in industrial environments. The source code used for the experiment was unknown to the subjects.

Independent Variables: We used two different software projects: Gantt Project and iTrust. Traceability was either available to a subject on a particular task (with traces) or it was not available (no traces). Traces were provided as they had been created by the original developers, some of them tracing to source code files and others to methods within files. 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. 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. Participants were not required to actually implement changes, but to identify the artifacts to be changed and to verbally describe each change. The oder in which tasks were assigned to subjects and the industry experience of a subject were also considered as variables.

Dependent Variables: This study had three dependent variables, the overall performance, the workflow, and the navigation strategy of subjects working on a maintenance task. Each variable has been operationalized by multiple measures. Performance was measured by the time to solve a task and the correctness of the solution. The workflow was measured as time that a subject spent on three different document types: the description of a task (task description), source code that was irrelevant for the solution of a task (irrelevant code), and source code that was relevant to solve a task (relevant code). The navigation strategy was measured by how often each of the four available navigation types were applied: 1) by double click on a node of the file tree, 2) by a click on a trace in the trace list, 3) by a click on a retrieved file in the results list of the search functionality, and 4) by navigating through tabs of open files.

Design: Participants were randomly assigned, but in a way that each had equally experienced each level of the independent variables project, task, and traceability.

Procedure and Material: We spent 20 min going step by step through the material, explaining the experiment and the two projects (Gantt, iTrust) to work on. The introduction further comprised instruction on 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. We allowed up to two hours for working on the eight assigned tasks. For each task, the participants had to capture their changes in the questionnaire that provided templates for capturing changes.

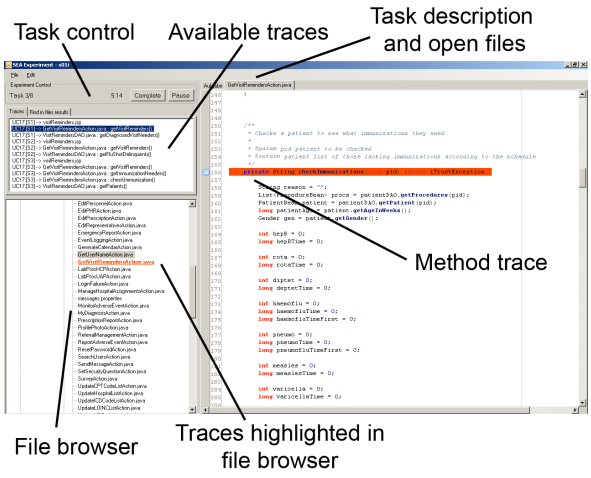

Figure 1. The experimenting tool and its major features

Experimenting Tool: In order to control and capture participants' actions during the experiment, we implemented a specific editor tool. We decided against a standard IDE (like Eclipse), because we could not ensure that all our subjects would be equally experienced in using the more sophisticated functionality of the tool, creating a possible bias in the results. The developed tool is a text editor with typical development support (see Figure 1). It provides a tree selector for browsing the project structure and opening files from it. Multiple files can be opened in separate tabs, and the content of files is syntax highlighted (both JAVA and JSP). Additionally, users can conveniently search within all files of a project and within a selected file. Traceability links are provided in a separate window above the tree selector. A click on a link opens the related file and shows the related method, if the trace refers to a specific method.

IV. Results

The discussion of results is split into three research questions: 1) how does the performance of a maintainer working with traceability differ from one working without traceability, 2) how much of the time to solve a task was spent on which type of document, and 3) what type of navigation has been used by subjects to solve a task. The following subsections refer to these questions respectively.

A. Performance with and without Traceability

We found that subjects working with traceability performed tasks on average 21% faster and created 60% more fully correct solutions. Statistically, both differences are highly significant. Subjects working on the Gantt project experienced a stronger support through traceability (24%

faster, 91% more correct solutions) than on the iTrust project (18% faster, 38% more correct solutions).

B. Workflows with and without traceability

Workflow refers to the amount of time that subjects spent on the document categories: 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).

Type of Task: We found that across all tasks, subjects working with traceability spent more time on relevant code than subjects working without traceability (45% Gantt A – 501% Gantt B). Subjects working without traceability spent 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.

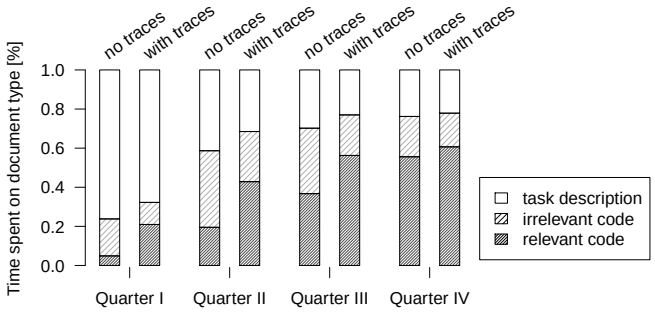

Figure 2. Percentages of time spent on understanding the task, browsing irrelevant code, and browsing relevant code separated into four quarters of progress per task (only correctly solved tasks)

Progress: Figure 2 shows an effort distribution across document types in relation to the stage of progress of a task solution. We divide the overall time, a subject was working on a task into four quarters of equal length. Only correct solutions have been aggregated for the figure. Independently of whether a task was performed with or without traceability, with each additional quarter, subjects spent a growing amount of time on relevant files. While the task description becomes at the same time less important. Especially in the first three quarters, subjects with traceability spent considerably more time on relevant files than those working without traceability. Indicating that subjects were willing to be led and explore the relevant places within the source code with traceability. The fourth quarter is distributed almost similar for subjects with and without traceability, indicating that the process of finding a solution to a task is not dependent on the availability of traceability. Subjects solving a task correctly with traceability spent across all quarters less time on the task description than subjects without traceability. For people, which are able to solve a task, traceability slightly lowers the time for understanding the problem.

Industry Experience: Figure 3 shows the effort distribution across document types in relation to the industry experience of subjects. For subjects working with traceability no significant differences are visible. For subjects working without traceability, the plot shows that more experienced subjects spent more time on relevant code and task description and less time on irrelevant code. Subjects with traceability and no industry experience create 159% more correct solutions, but that effect is diminishing for subjects with more experience (4–6 years of experience: 6%).

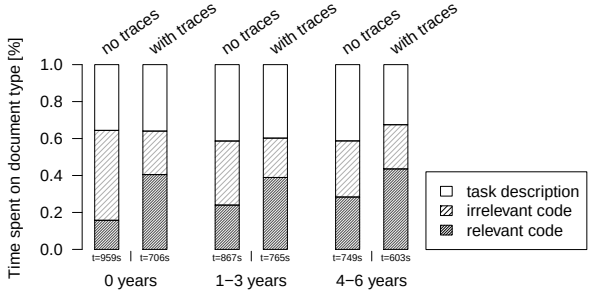

Figure 3. Percentages of time spent on understanding the task, browsing irrelevant code, and browsing relevant code in relation to the industry experience of subjects in years

C. Navigation strategies with and without traceability

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 (trace).

Task Type: For tasks were traceability was available, it was used for $20-62\%$ of the navigations depending on the task. Subjects working without traceability replace the missing trace navigation with searches ($34-48\%$ of the navigations). For tasks with traceability, the search functionality is used to a much smaller extend ($1-23\%$). Navigation from the file tree has been used by subjects without traceability for $16-30\%$ and by subjects with traceability for $4-23\%$ of all navigations. The availability of traceability seems to have no effect on that type of navigation. Subjects without traceability navigated $26-38\%$ through tabs and subjects with traceability $18-52\%$. By further evaluating the data, we found that subjects regularly used traces also to focus files which were open already. That fact explains the lower number of tab navigations.

Progress: Figure 4 shows the distribution of navigation types in relation to the progress of a task solution (four quarters), without and with traceability and includes only correctly solved tasks. Subjects working with traceability used it in quarter one extensively. In the following quarters traceability becomes stepwise less relevant and is mostly replaced by tab navigation. Subjects without traceability,

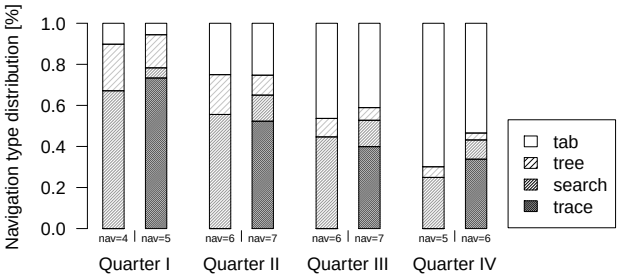

Figure 4. Distribution of navigation types as percentage of all navigations separated into four quarters of progress per correctly solved task

apply the search functionality instead, also with a shrinking relevance from quarter to quarter. In quarters three and four, subjects with traceability apply less tab navigation than those without traceability. We found that subjects used traces to switch tabs, instead of finding and selecting the tab directly.

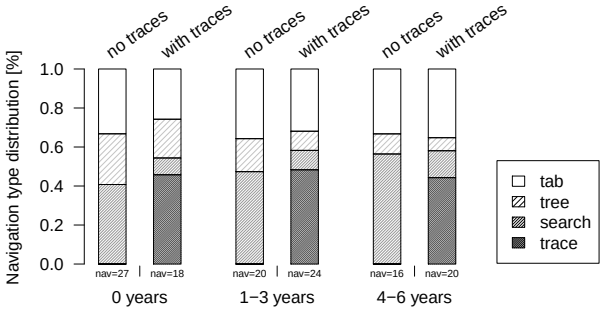

Figure 5. Distribution of employed navigation types as percentage of all navigations in relation to the industry experience of subjects

Industry Experience: Figure 5 shows the distribution of navigation types in relation to the industry experience of the subjects that were performing the task. There is a clear tendency, independent of available traceability, that more experienced subjects use searches more frequently for navigating. These searches replace tree navigation, which is more frequently used by less experienced subjects. Traceability navigation appears to largely displace search-based navigation and used equally well independent of experience.

V. Threats to Validity

This section discusses what is considered to be the most important threats to the validity of the experiment. 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. In order to exhaustively explore these effects an additional study with more experienced subjects is required and planned by the authors. We tried to keep all aspects of the experiment as realistic as possible, applying two systems, using four tasks per project and having different kinds of real tasks (bug reports vs. feature request). Systems, tasks and traces have been used in the original state. Focusing on the tasks we selected, our results show that all are neither overly easy nor unsolvable, suggesting a balanced selection. To decrease variability in knowledge across participants we provided a written introductory tutorial. 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.

VI. Conclusions

We conducted a controlled experiment with 52 subjects performing 315 maintenance tasks on two third-party development projects: half of them with and the other half without traceability navigation. Our finding is that traceability navigation has a profound effect on the performance, the quality, and the workflow of how change tasks are tackled. We found that subjects relied predominantly on traceability navigation when it was available, displacing mostly the search navigation which was predominant when traceability navigation was not available. Participants adopted traceability immediately, from the first performed task as their major way of navigation within the source code (without training). Traceability navigation was the means to quickly identifying which parts of the code need changing. Understanding the change task appeared to benefit little from traceability and neither did formulating a solution once the relevant code was found. Concluding, our study gives additional rationale to justify significant effort in traceability research.

References

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

[2] 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.

[3] 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.

[4] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE TSE*, vol. 32, pp. 971–987, 2006.

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

[6] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. 1st Int'l Conf. Req. Eng. ICRE94.*, 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.