ICIC Express Letters
Part B: Applications
Volume **7**, Number **11**, November **2016**

ICIC International ©2016 ISSN 2185-2766
pp. **2347–2352**

STUDY ON RECOVERING TRACE LINKS AMONG SOFTWARE ARTIFACTS

Jinshui Wang[1,2], Chia-Jung Lee[1,2] and Xingsi Xue[1,2]

[1]College of Information Science and Engineering
[2]Fujian Provincial Key Laboratory of Big Data Mining and Applications
Fujian University of Technology
No. 3, Xueyuan Road, University Town, Minhou, Fuzhou 350118, P. R. China
{ ymkscom; leecj2009; xxs }@gmail.com

Received April 2016; accepted July 2016

Abstract. *Developers often have to trace links among software artifacts during software maintenance and evolution tasks. This activity, which is referred to as traceability recovery in software engineering, is human-intensive and knowledge-intensive. However, little is still known on the process of traceability recovery. Our paper presents an exploratory study conducted with 8 subjects that require recovering traceability between different artifacts for an open source software system in a controlled environment. Our study yields surprising observations as follows: 1) comparing with the difficulty of the task, the familiarity with subject system and task could be the key factor that affects the effort of recovering traceability links; 2) recovering requirements-to-code traces is more difficult than recovering other types of traces.*

Keywords: Traceability recovery, Software maintenance, Exploratory experiment, Human study

1. **Introduction.** Traceability, which is defined as the ability to trace the relationships between software artifacts, has been recognized as a significant contributor to efficient software and system quality [1]. Traceability is shown to be useful in many software maintenance and evolution tasks [1,2], such as system verification and validation (V&V), change management, reuse analysis, program comprehension, regression testing and regulatory compliance. Being considered as the essential means to ensure that the source code of a software system is consistent with its requirements and that all and only the specified requirements have been implemented [3], traceability is a necessary component of the approval and certification process in most safety-critical systems [4]. Furthermore, traceability is also considered as a best practice and appointed by many major engineering standards (e.g., ISO 15504) and organizations (e.g., CMMI Product Team 2010) [5].

In practice, traceability information is typically presented and visualized using the requirements traceability matrix (RTM), which is able to document bi-directional traces between requirement and other software artifacts. Due to the complexity of software systems and the cross-cutting concerns of requirement distributed in different artifacts, the process of creating and maintaining an RTM is time-consuming and error-prone [1]. To address this problem, various techniques have been presented to provide (semi-) automatic assistance in traceability tasks, using information retrieval (IR) [6], static analysis [7,8], and dynamic analysis [9]. Researchers have empirically shown that these proposals can reduce developer's effort in tracing links between different artifacts and improve the quality of traceability recovery results.

In spite of the success of these techniques, traceability recovery remains a human-intensive and knowledge-intensive activity, and it is still risky to neglect human factors in the process [10]. However, little is still known on the process of traceability recovery. Due to the lack of knowledge that reflects practices of traceability experts and practitioners

[11], few software companies will choose to implement traceability processes [12]. A better understanding of how people recover trace links is essential for the researchers who wish to improve their existing techniques, and it is also needed by those tool developers and practitioners who provide trace recovery features and face the challenges of planning and managing trace recovery activities in industrial practice, respectively.

Researchers have conducted case studies and experiments to identify the relations between tracing results, and the key aspects of traceability recovery process, for example, the effort of recovering traceability links [13], the presence of automated methods or tools that provide suggestions for a specific task [14], and different external factors (e.g., human factors, task properties and in-process feedbacks) [10]. However, they did not identify the key factor in affecting the effort of recovering traceability. Furthermore, their experiment only concentrated on traceability links between requirements and code, and left out other types of traceability links (e.g., bugs and code, requirement and bug).

In this paper, we focus on understanding how developers trace links between different artifacts. More precisely, we try to answer the following two important research questions.

RQ 1. What is the key factor in affecting the effort of recovering traceability links?

RQ 2. Does the type of traceability links impact trace recovery effort?

To this end, we conduct an exploratory study of traceability recovery. In particular, we recruited 8 undergraduate students from our school, and each of them is given an unfamiliar system (JEdit) and asked to work on five traceability recovery tasks. After that, we analyze the screen-recorded videos of each participant's tracing process and conduct post-experiment questionnaires and interviews with the participants.

The remainder of the paper is organized as follows. Section 2 describes the design of our exploratory study. Section 3 presents the results of our experimental study and Section 4 concludes the paper with a summary of our findings.

2. **Experiment Design.** Our exploratory study of developers performing traceability recovery is based on JEdit, which is an open source Java system. Before the experiment, we introduce to the participants the background and relevant domain knowledge about the subject system. To ensure all participants' understandings of their tasks are similar, we also introduce the tasks to be completed and demonstrate the typical usage scenarios(s) of the involved requirements. During the experiment, we ask one assistant to help the participants realize the tasks and configure the environment. Table 1 summarizes the subject system.

Table 1. Subject systems, revisions, bugs and feature requests

Revision	Source code		Bug		Feature request	
	Classes	Methods	Closed	Open&Pending	Closed	Open&Pending
4480	1055	7401	3700	230	294	203

During the procedure of the experiment, all 8 participants are respectively given five traceability recovery tasks. Based on an overall consideration of complexity of the subject system, participants' familiarity with the system, and the representation of traceability links, we select the following five requirements.

1) Set the foreground color when text is selected
2) Search and replace text content
3) Highlight syntax token
4) Autosave turned off for untitled documents
5) Auto indent for Java

The difficulty of these requirements is moderate which ensures they can be easily understood by the participants. Note that our recent study [15] suggests that developers often feel difficult to formulate a suitable query (e.g., keywords) from feature/requirement

descriptions, which is a crucial factor in traceability recovery. To observe the difference in search behavior, these requirements were described in Chinese, and participants had to conceive queries before starting with code search.

For each task, the participants are requested to identify as many software artifacts that they deemed to be relevant to the given requirement as possible. To help the participants get familiar with the assigned tasks, we offer an example before the experiment began. In addition, all necessary software (e.g., integrated development environment, and SVN client) are installed and configured ahead of time, so that the participants can concentrate on their tasks.

In order to analyze the actions and processes of each participant in completing the assigned tasks, all the participants are required to run a full-screen recorder once they start the tasks. Furthermore, the participants are also asked to document their traceability recovery results according to the given template, and submitted them after the experiment. Finally, all the participants are asked to take part in a post-study questionnaire and an interview to provide more information which is difficult to learn by analyzing the screen-recorded videos of their traceability recovery process.

3. **Experiment Results.** Our experiment produced 11 hours 47 minutes of full-screen videos of 8 participants' work on 5 traceability recovery tasks on JEdit. Based on these videos, we analyze each participant's tracing processes as follows. First, we identify the moments when the participant begins, finishes or switches the tasks, so that the time spent in each task can be calculated. Furthermore, because there are four types of traceability links (i.e., requirement to source code, requirement to revision, requirement to bug report, and requirement to feature request) to be recovered, we also record the time spent in recovering each type of traceability links.

The key factor affecting effort (*RQ 1*). Let us first investigate the overall time distributions of each task as shown in Table 2. Note that the time consumption varies greatly for different tasks or participants. For example, the first participant (T1) spends 4731 seconds for task 1, while spends 1029 seconds for task 5. Besides, for task 1, the first participant (T1) needs 4731 seconds to complete the job, while the sixth participant (T6) only needs 448 seconds. Among these 8 participants, 5 participants need the most time on Task 1.

To explore the impact of task property on time distributions, all participants are required to choose the most difficult and easiest task in a post-study questionnaire. The results of the questionnaire show that no participant considers Task 1 as the most difficult one. On the contrary, 3 out of 8 participants consider it as the easiest task. Based on the screen videos of the participants' work and the post-experiment interview, we find out the leading cause behind the statistic. These traceability recovery tasks and the subject

Table 2. The amount of time spent in each task (seconds)

	Task1	Task2	Task3	Task4	Task5
T1	4731	1168	1899	1194	1029
T2	2687	513	1074	766	925
T3	1925	670	385	338	450
T4	998	279	856	451	369
T5	1250	1171	815	473	821
T6	448	1621	2110	1179	895
T7	955	653	602	1256	1024
T8	1160	1525	416	540	789
Avg	1759.25	950	1019.63	774.63	787.75

TABLE 3. The sequence of tasks performed by each participant

	Task1	Task2	Task3	Task4	Task5
T1	1	2	3	4	5
T2	1	2	3	4	5
T3	1	2	3	4	5
T4	5	1	2	3	4
T5	1	2	3	4	5
T6	5	1	2	3	4
T7	1	2	3	4	5
T8	1	2	3	4	5

system (JEdit) are unfamiliar to most participants, and may be difficult to learn. Therefore, it took participants a lot of time to become familiar with the system and tasks, leading most participants to spend the largest portion of their time on the first task. The sequence of tasks performed by each participant is summarized in Table 3. From Table 2 and Table 3, we could conclude that the familiarity of subject system and task rather than the difficulty of task is the decisive factor that affects the effort of participant.

Type of traces and effort (*RQ 2*). Regarding this research question, we tried to understand which type of trace links requires more effort to recover. Table 4 shows that all participants devoted the most time to recovering requirements-to-code traces. Among these eight participants, seven spend more than half their time on recovering requirements-to-code traces, only one spends relatively less time (45.5% of his time) on it. It is intuitive to assume that recovering requirements-to-code traces requires more effort since source code is more complex than other artifacts. The post-study questionnaire data shows that recovering requirements-to-code traces are considered the hardest by seven participants, while only one participant considered recovering requirements-to-bug is the hardest. Meanwhile, recovering requirements-to-bugs is considered the easiest by five participants, while three participants considered recovering requirements-to-features is the easiest.

Through the post-experiment interview with the participants, two key factors which may have an impact on trace effort were identified. First, the complexity of artifacts to be analyzed is of great significance. In particular, participants are required to recover requirements-to-code traces in a harsh scenario of unfamiliar tasks without automatic tool support. During the procedure of experiment, individual participant could only investigate a small set of program elements due to the limit of energy and time, which leads to a shortage of understanding of the system and tasks. Furthermore, in order to be consistent with the industrial setting [13], no source code documents but a few comments are

TABLE 4. The amount of time spent in different types of traces (seconds)

	requirement-code	requirement-revision	requirement-bug	requirement-feature
T1	8000	1245	320	456
T2	3093	1578	591	703
T3	2045	826	542	355
T4	1712	818	294	129
T5	2061	1218	574	677
T6	3522	1827	604	340
T7	2395	1311	674	110
T8	2658	721	505	546
Avg	3185.75	1193	513	414.5

provided for the participants. Under these circumstances, most participants found it is difficult to comprehend the source code, and they have to spend more effort on recovering requirements-to-code traces. Second, compared with the source code, the syntax of other types of software artifacts (e.g., bug reports, revision comments) is closer to the nature language. Therefore, it is easier to identify textual keywords that participants perceived to be relevant to the requirement, based on, for example, feature requests or bug reports. In addition, the search tools often return many results by using these identified keywords. It is more difficult for participants to read and determine whether the code was relevant to the given tasks than other artifacts.

This observation suggests that recovering requirements-to-code traces is more difficult than other types of traces, because understanding the purpose of program elements (e.g., class or method) as a whole is more difficult than understanding the textual description in other artifacts. This also indicates that helping developers to explore and to understand codes has been an important challenge in software engineering research. The lack of automatic traceability recovery mechanisms increases the difficulty in comprehending, exploring, and capturing program elements. Understanding the difficulty behind traceability recovery would allow it to provide more contextual-sensitive support for what developers are currently working on.

4. **Conclusion.** In this paper, we report an exploratory study of traceability recovery process in a controlled environment, involving 8 participants and 5 traceability recovery tasks for an open source software system. Our study reveals several interesting facts about the traceability recovery process: 1) comparing with the difficulty of the task, the familiarity with subject system and task could be the key factor that affects the effort of recovering traceability links; 2) recovering requirements-to-code traces is more difficult than recovering other types of traces. Future work will investigate more precisely the relationship and difference between traceability recovery process and feature location process.

Acknowledgment. This work is supported by the National Natural Science Foundation of China (Nos. 61402108, 61503082), Foundation for Scientific Research of Fujian Education Committee (Nos. JA15348, JA15336, JA13211), Research and Development Program of Fujian University of Technology (Nos. GY-Z15101, GY-Z15121). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, E. A. Holbrook, S. Vadlamudi and A. April, Requirements tracing on target (retro): Improving software maintenance through traceability recovery, *Innovations in Systems and Software Engineering*, vol.3, no.3, pp.193-202, 2007.

[2] S. Nair, D. L. V. J. Luis and S. Sen, A review of traceability research at the requirements engineering conference$^{RE@21}$, *Proc. of the 21st IEEE International Requirements Engineering Conference*, Rio de Janeiro, Brazil, pp.222-229, 2013.

[3] N. Ali, Y. G. Guéhéneuc and G. Antoniol. Trustrace: Mining software repositories to improve the accuracy of requirement traceability links, *IEEE Trans. Software Engineering*, vol.39, no.39, pp.725-741, 2013.

[4] J. Cleland-Huang, O. C. Gotel, J. H. Hayes, P. Der and A. Zisman, Software traceability: Trends and future directions, *On Future of Software Engineering*, Hyderabad, India, pp.55-69, 2014.

[5] A. Egyed, S. Biffl, M. Heindl and P. Grunbacher, Determining the cost-quality trade-off for automated software traceability, *Proc. of the 20th IEEE/ACM International Conference on Automated Software Engineering*, New York, USA, pp.360-363, 2005.

[6] C. Liu, G. Lai and X. Wang, Analysis and improvement on retrieval methods for traceability links between source code and documentation, *Tien Tzu Hsueh Pao/Acta Electronica Sinica*, vol.37, no.S1, pp.22-30, 2009.

[7] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia and D. Binkley, Recovering test-to-code traceability using slicing and textual analysis, *Journal of Systems and Software*, vol.88, no.2, pp.147-168, 2014.

[8] H. Kuang, P. Mader, H. Hu, A. Ghabi, L. Huang, J. Lu and A. Egyed, Can method data dependencies support the assessment of traceability between requirements and source code? *Journal of Software: Evolution and Process*, vol.27, no.11, pp.838-866, 2015.

[9] B. Dit, M. Revelle and D. Poshyvanyk, Integrating information retrieval, execution and link analysis algorithms to improve feature location in software, *Empirical Software Engineering*, vol.18, no.2, pp.277-309, 2013.

[10] J. Wang, Z. Xing, W. Zhao and X. Peng, How developers perform feature location tasks: A human-centric and process-oriented exploratory study, *Journal of Software: Evolution and Process*, vol.25, no.11, pp.1193-1224, 2013.

[11] R. Oliveto, G. Antoniol, A. Marcus and J. Hayes, Software artefact traceability: The never-ending challenge, *Proc. of the 23rd IEEE International Conference on Software Maintenance*, Paris, France, pp.485-488, 2007.

[12] G. Regan, F. Mccaffery, K. Mcdaid and D. Flood, Traceability-why do it? *Software Process Improvement and Capability Determination*, Springer, pp.161-172, 2012.

[13] A. Egyed, F. Graf and P. Grunbacher, Effort and quality of recovering requirements-to-code traces: Two exploratory experiments, *Proc of the 18th IEEE International Requirements Engineering Conference*, Sydney, NSW, Australia, pp.221-230, 2010.

[14] D. Cuddeback, A. Dekhtyar and J. Hayes. Automated requirements traceability: The study of human analysts, *Proc. of the 18th IEEE International Requirements Engineering Conference*, Sydney, NSW, Australia, pp.231-240, 2010.

[15] J. Wang, X. Peng, Z. Xing and W. Zhao, Improving feature location practice with multi-faceted interactive exploration, *Proc. of the 35th International Conference on Software Engineering*, San Francisco, USA, pp.762-771, 2013.